A Modular Splitting Framework for Saturation Theorem Proving

Ghilain Bergeron, Florent Krasnopol and Sophie Tourret

September 9, 2026

Abstract

We formalize in Isabelle/HOL a framework for splitting [2], a theorem proving technique that extends saturation-based calculi with branching abilities. The framework preserves the completeness of the original calculus. We focus here on the simplest splitting model described in details in the first three sections of "Unifying Splitting" by Gabriel Ebner, Jasmin Blanchette and Sophie Tourret [3] and provide an extension of the ordered resolution calculus with a variant of splitting called Lightweight AVATAR. A paper describing the present formalization has been accepted at ITP'25 [1].

Contents

1 Limsup of Lazy Lists **2**
1.1 Library . . . 2
1.2 Infimum . . . 2
1.3 Infimum up-to . . . 3
1.4 Limsup . . . 4
1.5 Limsup up-to . . . 5

2 Disjunctive Consequence Relations **8**

3 Extension to Negated Formulas **9**

4 Calculi and Derivations **11**

5 Annotated Formulas and Consequence Relations **14**

6 Lifting Calculi to Add Annotations **16**
6.1 Local saturation . . . 26

7 Lifting to Non-ground Calculi **27**
7.1 Standard Lifting 28
7.2 Strong Standard Lifting 29
7.3 Lifting with a Family of Tiebreaker Orderings 29
7.4 Lifting with a Family of Redundancy Criteria 32

8 Core splitting calculus **36**
8.1 The inference rules 37
8.2 The redundancy criterion 38
8.3 Standard completeness 40
8.4 Strong completeness 41

9 Extensions: Inferences and simplifications **42**
9.1 Simplifications 42
9.1.1 The Split Rule 43
9.1.2 The Collect Rule 47
9.1.3 The Trim Rule 48
9.2 Extra Inferences 50
9.2.1 The StrongUnsat Rule 51
9.2.2 The Tauto Rule 52
9.2.3 The Approx Rule 53
9.3 Combining all simplifications and optional inferences 55
9.4 The BinSplit Simplification Rule 56
9.5 Ordered Resolution with a Disjunctive Consequence Relation 58
9.6 Lightweight Avatar without BinSplit 61
9.7 Lightweight Avatar 62

1 Limsup of Lazy Lists

theory *Lazy-List-Limsup*
imports *Coinductive.Coinductive-List*
begin

1.1 Library

lemma *less-llength-ltake*: $i < llength\ (ltake\ k\ Xs) \longleftrightarrow i < k \wedge i < llength\ Xs$
⟨*proof*⟩

1.2 Infimum

definition *Inf-llist* :: ‹*'a set llist* ⇒ *'a set*› **where**
‹*Inf-llist Xs* = $(\bigcap i \in \{i.\ enat\ i < llength\ Xs\}.\ lnth\ Xs\ i)$›

lemma *Inf-llist-subset-lnth*: ‹*enat i* < *llength Xs* ⟹ *Inf-llist Xs* ⊆ *lnth Xs i*›
⟨*proof*⟩

lemma *Inf-llist-imp-exists-index*:
assumes ‹¬ *lnull Xs*›
shows ‹*x* ∈ *Inf-llist Xs* ⟹ ∃ *i*. *enat i* < *llength Xs* ∧ *x* ∈ *lnth Xs i*›
⟨*proof*⟩

lemma *Inf-llist-imp-all-index*: ‹*x* ∈ *Inf-llist Xs* ⟹ ∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i*›
⟨*proof*⟩

lemma *all-index-imp-Inf-llist*: ‹∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i* ⟹ *x* ∈ *Inf-llist Xs*›
⟨*proof*⟩

lemma *Inf-llist-LNil*[*simp*]: ‹*Inf-llist LNil* = *UNIV*›
⟨*proof*⟩

lemma *Inf-llist-LCons*[*simp*]: ‹*Inf-llist* (*LCons X Xs*) = *X* ∩ *Inf-llist Xs*›
⟨*proof*⟩

lemma *lhd-subset-Inf-llist*: ‹¬ *lnull Xs* ⟹ *Inf-llist Xs* ⊆ *lhd Xs*›
⟨*proof*⟩

1.3 Infimum up-to

definition *Inf-upto-llist* :: ‹'*a set llist* ⇒ *enat* ⇒ '*a set*› **where**
‹*Inf-upto-llist Xs j* = (⋂ *i* ∈ {*i*. *enat i* < *llength Xs* ∧ *enat i* ≤ *j*}. *lnth Xs i*)›

lemma *Inf-upto-llist-eq-Inf-llist-ltake*: ‹*Inf-upto-llist Xs j* = *Inf-llist* (*ltake* (*eSuc j*) *Xs*)›
⟨*proof*⟩

lemma *Inf-upto-llist-enat-0*[*simp*]:
‹*Inf-upto-llist Xs* (*enat 0*) = (*if lnull Xs then UNIV else lhd Xs*)›
⟨*proof*⟩

lemma *Inf-upto-llist-Suc*[*simp*]:
‹*Inf-upto-llist Xs* (*enat* (*Suc j*)) =
Inf-upto-llist Xs (*enat j*) ∩ (*if enat* (*Suc j*) < *llength Xs then lnth Xs* (*Suc j*) *else UNIV*)›
⟨*proof*⟩

lemma *Inf-upto-llist-infinity*[*simp*]: ‹*Inf-upto-llist Xs* ∞ = *Inf-llist Xs*›
⟨*proof*⟩

lemma *Inf-upto-llist-0*[*simp*]: ‹*Inf-upto-llist Xs 0* = (*if lnull Xs then UNIV else lhd Xs*)›
⟨*proof*⟩

lemma *Inf-upto-llist-eSuc*[*simp*]:

‹*Inf-upto-llist Xs* (*eSuc j*) =
(*case j of*
enat k ⇒ *Inf-upto-llist Xs* (*enat* (*Suc k*))
| ∞ ⇒ *Inf-llist Xs*)›
⟨*proof*⟩

lemma *Inf-upto-llist-anti*[*simp*]: ‹*j* ≤ *k* ⟹ *Inf-upto-llist Xs k* ⊆ *Inf-upto-llist Xs j*›
⟨*proof*⟩

lemma *Inf-llist-subset-Inf-upto-llist*: ‹*Inf-llist Xs* ⊆ *Inf-upto-llist Xs j*›
⟨*proof*⟩

lemma *elem-Inf-llist-imp-Inf-upto-llist*:
‹*x* ∈ *Inf-llist Xs* ⟹ *x* ∈ *Inf-upto-llist Xs* (*enat j*)›
⟨*proof*⟩

lemma *Inf-upto-llist-subset-lnth*: ‹*j* < *llength Xs* ⟹ *Inf-upto-llist Xs j* ⊆ *lnth Xs j*›
⟨*proof*⟩

lemma *Inf-llist-imp-Inf-upto-llist*:
assumes ‹*X* ⊆ *Inf-llist Xs*›
shows ‹*X* ⊆ *Inf-upto-llist Xs* (*enat k*)›
⟨*proof*⟩

1.4 Limsup

definition *Limsup-llist* :: ‹*'a set llist* ⇒ *'a set*› **where**
‹*Limsup-llist Xs* =
(⋂ *i* ∈ {*i*. *enat i* < *llength Xs*}. ⋃ *j* ∈ {*j*. *i* ≤ *j* ∧ *enat j* < *llength Xs*}. *lnth Xs j*)›

lemma *Limsup-llist-LNil*[*simp*]: ‹*Limsup-llist LNil* = *UNIV*›
⟨*proof*⟩

lemma *Limsup-llist-LCons*:
‹*Limsup-llist* (*LCons X Xs*) = (*if lnull Xs then X else Limsup-llist Xs*)› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *lfinite-Limsup-llist*: ‹*lfinite Xs* ⟹ *Limsup-llist Xs* = (*if lnull Xs then UNIV else llast Xs*)›
⟨*proof*⟩

lemma *Limsup-llist-ltl*: ‹¬ *lnull* (*ltl Xs*) ⟹ *Limsup-llist Xs* = *Limsup-llist* (*ltl Xs*)›
⟨*proof*⟩

lemma *Inf-llist-subset-Limsup-llist*: ‹*Inf-llist Xs* ⊆ *Limsup-llist Xs*›
⟨*proof*⟩

lemma *image-Limsup-llist-subset*: ‹*f* ‘ *Limsup-llist Ns* ⊆ *Limsup-llist* (*lmap* ((‘) *f*) *Ns*)›
⟨*proof*⟩

lemma *Limsup-llist-imp-exists-index*:
assumes ‹¬ *lnull Xs*›
shows ‹*x* ∈ *Limsup-llist Xs* ⟹ ∃ *i*. *enat i* < *llength Xs* ∧ *x* ∈ *lnth Xs i*›
⟨*proof*⟩

lemma *finite-subset-Limsup-llist-imp-exists-index*:
assumes
nnil: ‹¬ *lnull Xs*› **and**
fin: ‹*finite X*› **and**
in-sup: ‹*X* ⊆ *Limsup-llist Xs*›
shows ‹∀ *i*. *enat i* < *llength Xs* ⟶ *X* ⊆ (⋃ *j* ∈ {*j*. *i* ≤ *j* ∧ *enat j* < *llength Xs*}. *lnth Xs j*)›
⟨*proof*⟩

lemma *Limsup-llist-lmap-image*:
assumes *f-inj*: ‹*inj-on f* (*Inf-llist* (*lmap g xs*))›
shows ‹*Limsup-llist* (*lmap* (λ*x*. *f* ‘ *g x*) *xs*) = *f* ‘ *Limsup-llist* (*lmap g xs*)› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *Limsup-llist-lmap-union*:
assumes ‹∀ *x* ∈ *lset xs*. ∀ *y* ∈ *lset xs*. *g x* ∩ *h y* = {}›
shows ‹*Limsup-llist* (*lmap* (λ*x*. *g x* ∪ *h x*) *xs*) =
Limsup-llist (*lmap g xs*) ∪ *Limsup-llist* (*lmap h xs*)› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *Limsup-set-filter-commute*:
assumes ‹¬ *lnull Xs*›
shows ‹*Limsup-llist* (*lmap* (λ*X*. {*x* ∈ *X*. *p x*}) *Xs*) = {*x* ∈ *Limsup-llist Xs*. *p x*}› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

1.5 Limsup up-to

definition *Limsup-upto-llist* :: ‹'*a set llist* ⇒ *enat* ⇒ '*a set*› **where**
‹*Limsup-upto-llist Xs k* =
(⋂ *i* ∈ {*i*. *enat i* < *llength Xs* ∧ *enat i* ≤ *k*}.
⋃ *j* ∈ {*j*. *i* ≤ *j* ∧ *enat j* < *llength Xs* ∧ *enat j* ≤ *k*}. *lnth Xs j*)›

lemma *Limsup-upto-llist-eq-Limsup-llist-ltake*:
‹*Limsup-upto-llist Xs j* = *Limsup-llist* (*ltake* (*eSuc j*) *Xs*)›
⟨*proof*⟩

lemma *Limsup-upto-llist-enat*[*simp*]:
‹*Limsup-upto-llist Xs* (*enat k*) =
(*if enat k* < *llength Xs then lnth Xs k else if lnull Xs then UNIV else llast Xs*)›
⟨*proof*⟩

lemma *Limsup-upto-llist-infinity*[*simp*]: ‹*Limsup-upto-llist Xs* ∞ = *Limsup-llist Xs*›
⟨*proof*⟩

lemma *Limsup-upto-llist-0*[*simp*]:
‹*Limsup-upto-llist Xs 0* = (*if lnull Xs then UNIV else lhd Xs*)›
⟨*proof*⟩

lemma *Limsup-upto-llist-eSuc*[*simp*]:
‹*Limsup-upto-llist Xs* (*eSuc j*) =
(*case j of*
enat k ⇒ *Limsup-upto-llist Xs* (*enat* (*Suc k*))
| ∞ ⇒ *Limsup-llist Xs*)›
⟨*proof*⟩

lemma *Liminf-upto-llist-imp-elem-Limsup-llist*:
assumes ‹∃ *i* < *llength Xs*. ∀ *j* ≥ *i*. *j* < *llength Xs* ⟶ *x* ∈ *Limsup-upto-llist Xs* (*enat j*)›
shows ‹*x* ∈ *Limsup-llist Xs*›
⟨*proof*⟩

end

theory *FSet-Extra*
imports *Main HOL−Library.FSet*
begin

This theory contains some additional lemmas regarding sets and finite sets, which were useful in the process of proving some lemmas in `Modular_Splitting_Calculus.thy` and `Lightweight_Avatar.thy`.

lemma *Union-of-singleton-is-setcompr*: ‹(⋃ *x* ∈ *A*. { *y*. *y* = *f x* ∧ *P x* }) = { *f x* | *x*. *x* ∈ *A* ∧ *P x* }›
⟨*proof*⟩

lemma *finite-because-singleton*: ‹(∀ *C1* ∈ *S*. ∀ *C2* ∈ *S*. *C1* = *C2*) ⟶ *finite S*›
for *S*
⟨*proof*⟩

lemma *finite-union-of-finite-is-finite*: ‹*finite E* ⟹ (∀ *D* ∈ *E*. *finite*({*f C* |*C*. *P C* ∧ *g C* = *D*})) ⟹
finite({*f C* |*C*. *P C* ∧ *g C* ∈ *E*})›

⟨*proof*⟩

definition *list-of-fset* :: *'a fset* ⇒ *'a list* **where**
‹*list-of-fset A* = (*SOME l*. *fset-of-list l* = *A*)›

lemma *fin-set-fset*: *finite A* ⟹ ∃ *Af*. *fset Af* = *A* ⟨*proof*⟩

lemma *fimage-snd-zip-is-snd* [*simp*]:
‹*length x* = *length y* ⟹ (λ(*x*, *y*). *y*) |‘| *fset-of-list* (*zip x y*) = *fset-of-list y*›
⟨*proof*⟩

lemma *if-in-ffUnion-then-in-subset*: ‹*x* |∈| *ffUnion A* ⟹ ∃ *a*. *a* |∈| *A* ∧ *x* |∈| *a*›
⟨*proof*⟩

lemma *fset-ffUnion-subset-iff-all-fsets-subset*: ‹*fset* (*ffUnion A*) ⊆ *B* ⟷ *fBall A* (λ *x*. *fset x* ⊆ *B*)›
⟨*proof*⟩

lemma *fBall-fset-of-list-iff-Ball-set*: ‹*fBall* (*fset-of-list A*) *P* ⟷ *Ball* (*set A*) *P*›
⟨*proof*⟩

lemma *wf-fsubset*: ‹*wfP* (|⊂|)›
⟨*proof*⟩

lemma *non-zero-fcard-of-non-empty-set*: ‹*fcard A* > *0* ⟷ *A* ≠ {||}›
⟨*proof*⟩

lemma *fimage-of-non-fempty-is-non-fempty*: ‹*A* ≠ {||} ⟹ *f* |‘| *A* ≠ {||}›
⟨*proof*⟩

lemma *Union-empty-if-set-empty-or-all-empty*:
‹*ffUnion A* = {||} ⟹ *A* = {||} ∨ *fBall A* (λ *x*. *x* = {||})›
⟨*proof*⟩

lemma *fBall-fimage-is-fBall*: ‹*fBall* (*f* |‘| *A*) *P* ⟷ *fBall A* (λ *x*. *P* (*f x*))›
⟨*proof*⟩

lemma *fset-map2*: ‹*v* ∈ *fset A* ⟹ *g* (*f v*) ∈ *set* (*map g* (*map f* (*list-of-fset A*)))›
⟨*proof*⟩

end

theory *Disjunctive-Consequence-Relations*
imports *Saturation-Framework.Calculus*
Propositional-Proof-Systems.Compactness
HOL−Library.Library
HOL−Library.Product-Lexorder

Lazy-List-Limsup
FSet-Extra
begin

2 Disjunctive Consequence Relations

no-notation *Sema.formula-semantics* (**infix** $\models$ *51*)

locale *consequence-relation* =
fixes
bot :: *'f* **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models$ *50*)
assumes
bot-entails-empty: $\{bot\} \models \{\}$ **and**
entails-reflexive: $\{C\} \models \{C\}$ **and**
entails-subsets: $M' \subseteq M \Longrightarrow N' \subseteq N \Longrightarrow M' \models N' \Longrightarrow M \models N$ **and**
entails-cut: $M \models N \cup \{C\} \Longrightarrow M' \cup \{C\} \models N' \Longrightarrow M \cup M' \models N \cup N'$ **and**
entails-compactness: $M \models N \Longrightarrow \exists\ M'\ N'.\ (M' \subseteq M \wedge N' \subseteq N \wedge$ *finite* $M' \wedge$ *finite* $N' \wedge M' \models N')$

begin

definition *order-double-subsets* :: (*'f set* ∗ *'f set*) ⇒ (*'f set* ∗ *'f set*) ⇒ *bool*
(**infix** $\preceq_s$ *50*) **where**
$\langle(\preceq_s) \equiv \lambda C1\ C2.\ fst\ C1 \subseteq fst\ C2 \wedge snd\ C1 \subseteq snd\ C2\rangle$
definition *order-double-subsets-strict* :: (*'f set* ∗ *'f set*) ⇒ (*'f set* ∗ *'f set*) ⇒ *bool*
(**infix** $\prec_s$ *50*) **where**
$\langle(\prec_s) \equiv \lambda C1\ C2.\ C1 \preceq_s C2 \wedge C1 \neq C2\rangle$

lemma *trivial-induction-order* : $\langle C1 \subseteq B \wedge C2 \subseteq B' \longrightarrow (C1,C2) \preceq_s (B,B')\rangle$
⟨*proof*⟩

lemma *zorn-relation-trans* : $\langle\forall\ C1\ C2\ C3.\ (C1 \preceq_s C2) \longrightarrow (C2 \preceq_s C3) \longrightarrow (C1 \preceq_s C3)\rangle$
⟨*proof*⟩

lemma *zorn-strict-relation-trans* :
$\langle\forall\,(C1::'f\ set \times\ 'f\ set)\ C2\ C3.\ (C1 \prec_s C2) \longrightarrow (C2 \prec_s C3) \longrightarrow (C1 \prec_s C3)\rangle$
⟨*proof*⟩

lemma *zorn-relation-antisym* : $\langle\forall\ C1\ C2.\ (C1 \preceq_s C2) \longrightarrow (C2 \preceq_s C1) \longrightarrow (C1 = C2)\rangle$
⟨*proof*⟩

lemma *entails-supsets* : $(\forall\ M'\ N'.\ (M' \supseteq M \wedge N' \supseteq N \wedge M' \cup N' = UNIV) \longrightarrow$

$M' \models N') \Longrightarrow M \models N$
⟨*proof*⟩

lemma *entails-each*: $M \models P \Longrightarrow \forall\, C{\in}M.\ N \models Q \cup \{C\} \Longrightarrow \forall\, D{\in}P.\ N \cup \{D\} \models Q \Longrightarrow N \models Q$
⟨*proof*⟩

lemma *entails-bot-to-entails-empty*: ‹$\{\} \models \{bot\} \Longrightarrow \{\} \models \{\}$›
⟨*proof*⟩

abbreviation *equi-entails* :: *'f set* ⇒ *'f set* ⇒ *bool* **where**
equi-entails M N ≡ $(M \models N \wedge N \models M)$

lemma *entails-cond-reflexive*: ‹$N \neq \{\} \Longrightarrow N \models N$›
⟨*proof*⟩

lemma *entails-empty-reflexive-dangerous*: ‹$\{\} \models \{\} \Longrightarrow M \models N$›
⟨*proof*⟩

definition *entails-conjunctive* :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models\cap$ *50*) **where**
$M \models\cap N \equiv \forall\, C{\in}N.\ M \models \{C\}$

sublocale *Calculus.consequence-relation* $\{bot\}$ ($\models\cap$)
⟨*proof*⟩
end

3 Extension to Negated Formulas

datatype *'a sign* = *Pos 'a* | *Neg 'a*

instance *sign* :: (*countable*) *countable*
⟨*proof*⟩

fun *neg* :: ‹*'a sign* ⇒ *'a sign*› **where**
‹*neg* (*Pos C*) = *Neg C*› |
‹*neg* (*Neg C*) = *Pos C*›

fun *to-V* :: *'a sign* ⇒ *'a* **where**
to-V (*Pos C*) = *C* |
to-V (*Neg C*) = *C*

lemma *neg-neg-A-is-A* [*simp*]: ‹*neg* (*neg A*) = *A*›
⟨*proof*⟩

fun *is-Pos* :: *'a sign* ⇒ *bool* **where**

is-Pos (*Pos C*) = *True* |
is-Pos (*Neg C*) = *False*

lemma *is-Pos-to-V*: ‹*is-Pos C* ⟹ *C* = *Pos* (*to-V C*)›
⟨*proof*⟩

lemma *is-Neg-to-V*: ‹¬ *is-Pos C* ⟹ *C* = *Neg* (*to-V C*)›
⟨*proof*⟩

lemma *pos-union-singleton*: ‹{*D*. *Pos D* ∈ *N* ∪ {*Pos X*}} = {*D*. *Pos D* ∈ *N*} ∪ {*X*}›
⟨*proof*⟩

lemma *tov-set*[*simp*]: ‹{*to-V C* |*C*. *to-V C* ∈ *A*} = *A*›
⟨*proof*⟩

lemma *pos-neg-union*: ‹{*P C* |*C*. *Q C* ∧ *is-Pos C*} ∪ {*P C* |*C*. *Q C* ∧ ¬ *is-Pos C*} = {*P C* |*C*. *Q C*}›
⟨*proof*⟩

context *consequence-relation*
begin
definition *entails-neg* :: *'f sign set* ⇒ *'f sign set* ⇒ *bool* (**infix** ⊨∼ *50*) **where**
entails-neg M N ≡ {*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *N*} ⊨ {*C*. *Pos C* ∈ *N*} ∪ {*C*. *Neg C* ∈ *M*}

lemma *swap-neg-in-entails-neg*: ‹{*neg A*} ⊨∼ {*neg B*} ⟷ {*B*} ⊨∼ {*A*}›
⟨*proof*⟩

lemma *ext-cons-rel*: ‹*consequence-relation* (*Pos bot*) *entails-neg*›
⟨*proof*⟩

interpretation *neg-ext-cons-rel*: *consequence-relation Pos bot entails-neg*
⟨*proof*⟩

lemma *pos-neg-entails-bot*: ‹{*C*} ∪ {*neg C*} ⊨∼ {*Pos bot*}›
⟨*proof*⟩

lemma *entails-of-entails-iff*:
‹{*C*} ⊨∼ *Cs* ⟹ *finite Cs* ⟹ *card Cs* ≥ *1* ⟹
(∀ *C*$_i$ ∈ *Cs*. *M* ∪ {*C*$_i$} ⊨∼ {*Pos bot*}) ⟹ *M* ∪ {*C*} ⊨∼ {*Pos bot*}›
⟨*proof*⟩

end

end

theory *Calculi-And-Annotations*
imports *Disjunctive-Consequence-Relations*
begin

4 Calculi and Derivations

context
begin

no-notation *Extended.extended.Pinf* (‹∞›)

typedef *'a infinite-llist* = ‹{ *l* :: *'a llist*. *llength l* = ∞ }›
morphisms *to-llist Abs-infinite-llist*
⟨*proof*⟩

setup-lifting *type-definition-infinite-llist*

lift-definition *llmap* :: ‹(*'a* ⇒ *'b*) ⇒ *'a infinite-llist* ⇒ *'b infinite-llist*› **is** *lmap*
⟨*proof*⟩

lift-definition *llnth* :: ‹*'a infinite-llist* ⇒ *nat* ⇒ *'a*› **is** *lnth* ⟨*proof*⟩

lift-definition *Liminf-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Liminf-llist*
⟨*proof*⟩

lift-definition *Limsup-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Limsup-llist*
⟨*proof*⟩

lift-definition *Sup-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Sup-llist* ⟨*proof*⟩

lift-definition *llhd* :: ‹*'a infinite-llist* ⇒ *'a*› **is** *lhd* ⟨*proof*⟩

lemma *llength-of-to-llist-is-infinite*: ‹*llength* (*to-llist L*) = ∞›
⟨*proof*⟩

end

locale *sound-inference-system* =
inference-system Inf + *sound-cons*: *consequence-relation bot entails-sound*
for
Inf :: *'f inference set* **and**
bot :: *'f* **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*)
+ **assumes**
sound: *ι* ∈ *Inf* ⟹ *set* (*prems-of ι*) ⊨*s* {*concl-of ι*}

no-notation *IArray.sub* (**infixl** !! *100*)

definition *is-derivation* :: $('f\ set \Rightarrow 'f\ set \Rightarrow bool) \Rightarrow ('f\ set\ infinite\text{-}llist) \Rightarrow bool$
where
$is\text{-}derivation\ R\ Ns \equiv \forall i.\ R\ (llnth\ Ns\ i)\ (llnth\ Ns\ (Suc\ i))$

definition *terminates* :: $'f\ set\ infinite\text{-}llist \Rightarrow bool$ **where**
$terminates\ Ns \equiv \exists i.\ \forall j{>}i.\ llnth\ Ns\ j = llnth\ Ns\ i$

abbreviation ‹*lim-inf* ≡ *Liminf-infinite-llist*›

abbreviation *limit* :: $'f\ set\ infinite\text{-}llist \Rightarrow 'f\ set$ **where** $limit\ Ns \equiv lim\text{-}inf\ Ns$

abbreviation *lim-sup* :: ‹$'f\ set\ infinite\text{-}llist \Rightarrow 'f\ set$› **where**
‹*lim-sup Ns* ≡ *Limsup-infinite-llist Ns*›

locale *calculus* = *inference-system Inf* + *consequence-relation bot entails*
for
$bot :: 'f$ **and**
Inf :: ‹$'f\ inference\ set$› **and**
entails :: $'f\ set \Rightarrow 'f\ set \Rightarrow bool$ (**infix** $\models$ *50*)
+ **fixes**
$Red_I :: 'f\ set \Rightarrow 'f\ inference\ set$ **and**
$Red_F :: 'f\ set \Rightarrow 'f\ set$
assumes
Red-I-to-Inf: $Red_I\ N \subseteq Inf$ **and**
Red-F-Bot: $N \models \{bot\} \Longrightarrow N - Red_F\ N \models \{bot\}$ **and**
Red-F-of-subset: $N \subseteq N' \Longrightarrow Red_F\ N \subseteq Red_F\ N'$ **and**
Red-I-of-subset: $N \subseteq N' \Longrightarrow Red_I\ N \subseteq Red_I\ N'$ **and**
Red-F-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_F\ N \subseteq Red_F\ (N - N')$ **and**
Red-I-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_I\ N \subseteq Red_I\ (N - N')$ **and**
Red-I-of-Inf-to-N: $\iota \in Inf \Longrightarrow concl\text{-}of\ \iota \in N \Longrightarrow \iota \in Red_I\ N$
begin

definition *saturated* :: $'f\ set \Rightarrow bool$ **where**
$saturated\ N \longleftrightarrow Inf\text{-}from\ N \subseteq Red_I\ N$

definition Red_I*-strict* :: $'f\ set \Rightarrow 'f\ inference\ set$ **where**
$Red_I\text{-}strict\ N = \{\iota.\ \iota \in Red_I\ N \vee (\iota \in Inf \wedge bot \in N)\}$

definition Red_F*-strict* :: $'f\ set \Rightarrow 'f\ set$ **where**
$Red_F\text{-}strict\ N = \{C.\ C \in Red_F\ N \vee (bot \in N \wedge C \neq bot)\}$

lemma *strict-calc-if-nobot*:
$\forall N.\ bot \notin Red_F\ N \Longrightarrow calculus\ bot\ Inf\ entails\ Red_I\text{-}strict\ Red_F\text{-}strict$
⟨*proof*⟩

definition *weakly-fair* :: $'f\ set\ infinite\text{-}llist \Rightarrow bool$ **where**
‹*weakly-fair Ns* ≡ *Inf-from* (*Liminf-infinite-llist Ns*) ⊆ *Sup-infinite-llist* (*llmap* Red_I *Ns*)›

abbreviation *fair* :: *'f set infinite-llist* ⇒ *bool* **where** *fair N* ≡ *weakly-fair N*

definition *derive* :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ▷ *50*) **where**
M ▷ *N* ≡ (*M* − *N* ⊆ *Red*$_F$ *N*)

lemma *derive-refl*: *M* ▷ *M* ⟨*proof*⟩

lemma *deriv-red-in*: ‹*M* ▷ *N* ⟹ *Red*$_F$ *M* ⊆ *N* ∪ *Red*$_F$ *N*›
⟨*proof*⟩

lemma *derive-trans*: *M* ▷ *N* ⟹ *N* ▷ *N'* ⟹ *M* ▷ *N'*
⟨*proof*⟩

end

locale *sound-calculus* = *sound-inference-system Inf bot entails-sound* +
consequence-relation bot entails
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*)
+ **fixes**
Red$_I$ :: *'f set* ⇒ *'f inference set* **and**
Red$_F$ :: *'f set* ⇒ *'f set*
assumes
Red-I-to-Inf: *Red*$_I$ *N* ⊆ *Inf* **and**
Red-F-Bot: *N* ⊨ {*bot*} ⟹ *N* − *Red*$_F$ *N* ⊨ {*bot*} **and**
Red-F-of-subset: *N* ⊆ *N'* ⟹ *Red*$_F$ *N* ⊆ *Red*$_F$ *N'* **and**
Red-I-of-subset: *N* ⊆ *N'* ⟹ *Red*$_I$ *N* ⊆ *Red*$_I$ *N'* **and**
Red-F-of-Red-F-subset: *N'* ⊆ *Red*$_F$ *N* ⟹ *Red*$_F$ *N* ⊆ *Red*$_F$ (*N* − *N'*) **and**
Red-I-of-Red-F-subset: *N'* ⊆ *Red*$_F$ *N* ⟹ *Red*$_I$ *N* ⊆ *Red*$_I$ (*N* − *N'*) **and**
Red-I-of-Inf-to-N: *ι* ∈ *Inf* ⟹ *concl-of ι* ∈ *N* ⟹ *ι* ∈ *Red*$_I$ *N*
begin

sublocale *calculus bot Inf entails*
⟨*proof*⟩
end

locale *statically-complete-calculus* = *calculus* +
assumes *statically-complete*: *saturated N* ⟹ *N* ⊨ {*bot*} ⟹ *bot* ∈ *N*
begin

lemma *inf-from-subs*: *M* ⊆ *N* ⟹ *Inf-from M* ⊆ *Inf-from N*
⟨*proof*⟩

lemma *nobot-in-Red*: ‹*bot* ∉ Red_F *N*›
⟨*proof*⟩

interpretation *strict-calculus*:
statically-complete-calculus bot Inf entails Red_I*-strict* Red_F*-strict*
⟨*proof*⟩

end

locale *dynamically-complete-calculus* = *calculus* +
assumes *dynamically-complete*:
‹*is-derivation* (▷) *Ns* ⟹ *fair Ns* ⟹ *llhd Ns* ⊨ {*bot*} ⟹ ∃ *i*. *bot* ∈ *llnth Ns i*›

5 Annotated Formulas and Consequence Relations

datatype ('*f*, '*v*::*countable*) *AF* = *Pair* (*F-of*: '*f*) (*A-of*: '*v sign fset*)

definition *is-interpretation* :: '*v sign set* ⇒ *bool* **where**
‹*is-interpretation J* = (∀ *v1*∈*J*. (∀ *v2*∈*J*. (*to-V v1* = *to-V v2* ⟶ *v1* = *v2*)))›

typedef '*v propositional-interpretation* = {*J* :: '*v sign set*. *is-interpretation J*}
⟨*proof*⟩

abbreviation *interp-of* ≡ *Abs-propositional-interpretation*
abbreviation *strip* ≡ *Rep-propositional-interpretation*

setup-lifting *type-definition-propositional-interpretation*

lift-definition *belong-to* :: '*v sign* ⇒ '*v propositional-interpretation* ⇒ *bool* (**infix** ∈$_J$ *90*)
is (∈)::('*v sign* ⇒ '*v sign set* ⇒ *bool*) ⟨*proof*⟩

definition *total* :: '*v propositional-interpretation* ⇒ *bool* **where**
‹*total J* ≡ (∀ *v*. (∃ v_J. v_J ∈$_J$ *J* ∧ *to-V* v_J = *v*))›

typedef '*v total-interpretation* = {*J* :: '*v propositional-interpretation*. *total J*}
⟨*proof*⟩

abbreviation *total-interp-of* ≡ (λ*x*. *Abs-total-interpretation* (*interp-of x*))
abbreviation *total-strip* ≡ (λ*x*. *strip* (*Rep-total-interpretation x*))

lemma *neg-notin-total-strip* [*simp*]: ‹(*neg a* ∉ *total-strip J*) = (*a* ∈ *total-strip J*)›
⟨*proof*⟩

lemma *neg-in-total-strip* [*simp*]: ‹(*neg a* ∈ *total-strip J*) = (*a* ∉ *total-strip J*)›

⟨*proof*⟩

setup-lifting *type-definition-total-interpretation*

lift-definition *belong-to-total* :: *'v sign* ⇒ *'v total-interpretation* ⇒ *bool* (**infix** $\in_t$ *90*)
  is $(\in_J)$::(*'v sign* ⇒ *'v propositional-interpretation* ⇒ *bool*) ⟨*proof*⟩

lemma *in-total-to-strip* [*simp*]: ‹$a \in_t J \longleftrightarrow a \in$ *total-strip* J›
  ⟨*proof*⟩

lemma *neg-prop-interp*: ‹(*v*::*'v sign*) $\in_J$ $J \Longrightarrow \neg$ ((*neg v*) $\in_J$ J)›
⟨*proof*⟩

lemma *neg-total-interp*: ‹(*v*::*'v sign*) $\in_t$ $J \Longrightarrow \neg$ ((*neg v*) $\in_t$ J)›
⟨*proof*⟩

lemma *neg-notin-total-interp*: ‹$\neg$ ($v \in_t J$) ⟹ ((*neg v*) $\in_t$ J)›
⟨*proof*⟩

definition *to-AF* :: *'f* ⇒ (*'f*, *'v*::*countable*) *AF* **where**
  ‹*to-AF C* = *Pair C* {||}›

lemma *F-of-to-AF*: ‹*F-of* (*to-AF* $\mathcal{C}$) = $\mathcal{C}$›
  ⟨*proof*⟩

lemma *A-of-to-AF*: ‹*A-of* (*to-AF* $\mathcal{C}$) = {||}›
  ⟨*proof*⟩

lemma *F-of-circ-to-AF-is-id* [*simp*]: ‹*F-of* ∘ *to-AF* = *id*›
  ⟨*proof*⟩

lemma *A-of-circ-to-AF-is-empty-set* [*simp*]: ‹*A-of* ∘ *to-AF* = (λ -. {||})›
  ⟨*proof*⟩

lemma *F-of-propositional-clauses* [*simp*]:
  ‹(∀ $x \in$ *set* $\mathcal{N}$. *F-of x* = *bot*) ⟹ *map F-of* $\mathcal{N}$ = *map* (λ -. *bot*) $\mathcal{N}$›
  ⟨*proof*⟩

lemma *F-of-Pair* [*simp*]: ‹*F-of* ∘ (λ(*x*, *y*). *AF.Pair x y*) = (λ(*x*, *y*). *x*)›
  ⟨*proof*⟩

lemma *A-of-Pair* [*simp*]: ‹*A-of* ∘ (λ(*x*, *y*). *AF.Pair x y*) = (λ(*x*, *y*). *y*)›
  ⟨*proof*⟩

lemma *map-A-of-map2-Pair*: ‹*length A* = *length B* ⟹ *map A-of* (*map2 AF.Pair A B*) = *B*›
⟨*proof*⟩

definition *Neg-set* :: *'v sign set* ⇒ *'v sign set* (∼- *55*) **where**
‹∼*V* ≡ {*neg v* |*v*. *v* ∈ *V*}›

definition *F-of-Inf* :: ((*'f*, *'v*::*countable*) *AF*) *inference* ⇒ *'f inference* **where**
‹*F-of-Inf* ι*AF* = (*Infer* (*map F-of* (*prems-of* ι*AF*)) (*F-of* (*concl-of* ι*AF*)))›

6 Lifting Calculi to Add Annotations

locale *calculus-with-annotated-consrel* = *sound-calculus bot Inf entails entails-sound* Red_I Red_F
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** |= *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** |=*s* *50*) **and**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set*
+ **fixes**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*›
assumes
fml-entails-C: ‹∀ *a* ∈ *asn C*. *sound-cons.entails-neg* {*map-sign fml a*} {*C*}›
and
C-entails-fml: ‹∀ *a* ∈ *asn C*. *sound-cons.entails-neg* {*C*} {*map-sign fml a*}›
and
asn-not-empty: ‹*asn C* ≠ {}›
begin

notation *sound-cons.entails-neg* (**infix** ‹|=*s*∼› *50*)

lemma *equi-entails-if-a-in-asns*: ‹*a* ∈ *asn C* ⟹ *a* ∈ *asn D* ⟹ {*C*} |=*s*∼ {*D*} ∧ {*D*} |=*s*∼ {*C*}›
⟨*proof*⟩

lemma *equi-entails-if-neg-a-in-asn*:
‹*a* ∈ *asn C* ⟹ *neg a* ∈ *asn D* ⟹ {*C*} |=*s*∼ {*neg D*} ∧ {*neg D*} |=*s*∼ {*C*}›
⟨*proof*⟩

definition ι*F-of* :: (*'f*, *'v*) *AF inference* ⇒ *'f inference* **where**
‹ι*F-of* ι = *Infer* (*List.map F-of* (*prems-of* ι)) (*F-of* (*concl-of* ι))›

definition *propositional-projection* :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ($proj_\bot$) **where**
‹$proj_\bot$ $\mathcal{N}$ = {$\mathcal{C}$. $\mathcal{C}$ ∈ $\mathcal{N}$ ∧ *F-of* $\mathcal{C}$ = *bot*}›

lemma *prop-proj-in*: ‹$proj_\bot$ $\mathcal{N}$ ⊆ $\mathcal{N}$›
⟨*proof*⟩

definition *enabled* :: (*'f*, *'v*) *AF* ⇒ *'v total-interpretation* ⇒ *bool* **where**

enabled $\mathcal{C}$ $J \equiv$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ (*total-strip* J)

lemma *subformula-of-enabled-formula-is-enabled*: ‹*A-of* $\mathcal{C}$ |⊂| *A-of* $\mathcal{C}'$ $\Longrightarrow$ *enabled* $\mathcal{C}'$ J $\Longrightarrow$ *enabled* $\mathcal{C}$ J›
⟨*proof*⟩

lemma *enabled-iff*: ‹*A-of* $\mathcal{C}$ = *A-of* $\mathcal{C}'$ $\Longrightarrow$ *enabled* $\mathcal{C}$ J $\longleftrightarrow$ *enabled* $\mathcal{C}'$ J›
⟨*proof*⟩

definition *enabled-set* :: ('*f*, '*v*) *AF set* ⇒ '*v total-interpretation* ⇒ *bool* **where**
‹*enabled-set* $\mathcal{N}$ J = ($\forall \mathcal{C} \in \mathcal{N}$. *enabled* $\mathcal{C}$ J)›

lemma *enabled-set-singleton* [*simp*]: ‹*enabled-set* {$\mathcal{C}$} J $\longleftrightarrow$ *enabled* $\mathcal{C}$ J›
⟨*proof*⟩

definition *enabled-inf* :: ('*f*, '*v*) *AF inference* ⇒ '*v total-interpretation* ⇒ *bool* **where**
‹*enabled-inf* ι J = ($\forall \mathcal{C} \in$ *set* (*prems-of* ι). *enabled* $\mathcal{C}$ J)›

definition *enabled-projection* :: ('*f*, '*v*) *AF set* ⇒ '*v total-interpretation* ⇒ '*f set* (**infix** *proj*$_J$ 60) **where**
‹$\mathcal{N}$ *proj*$_J$ J = {*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C} \in \mathcal{N} \wedge$ *enabled* $\mathcal{C}$ J}›

lemma *Union-of-enabled-projection-is-enabled-projection*: ‹($\bigcup$ $\mathcal{C} \in \mathcal{N}$. {$\mathcal{C}$} *proj*$_J$ $\mathcal{J}$) = $\mathcal{N}$ *proj*$_J$ $\mathcal{J}$›
⟨*proof*⟩

lemma *projection-of-enabled-subset*:
‹*fset* $B \subseteq$ *total-strip* J $\Longrightarrow$ {*AF.Pair* C (A |∪| B)} *proj*$_J$ J = {*AF.Pair* C A} *proj*$_J$ J›
⟨*proof*⟩

lemma *Un-of-enabled-projection-is-enabled-projection-of-Un*:
‹($\bigcup$ x. P x) *proj*$_J$ J = ($\bigcup$ x. P x *proj*$_J$ J)›
⟨*proof*⟩

lemma *enabled-projection-of-Int-is-Int-of-enabled-projection*:
‹$x \in (\bigcap$ S) *proj*$_J$ J $\Longrightarrow$ $x \in \bigcap$ { x *proj*$_J$ J | x. $x \in S$ }›
⟨*proof*⟩

definition *enabled-projection-Inf* :: ('*f*, '*v*) *AF inference set* ⇒ '*v total-interpretation* ⇒
'*f inference set* (**infix** *ιproj*$_J$ 60) **where**
‹I *ιproj*$_J$ J = {*ιF-of* ι | ι. $\iota \in I \wedge$ *enabled-inf* ι J}›

fun *fml-ext* :: '*v sign* ⇒ '*f sign* **where**
fml-ext (*Pos* v) = *Pos* (*fml* v) |
fml-ext (*Neg* v) = *Neg* (*fml* v)

lemma *fml-ext-is-mapping*: ‹*fml-ext v = map-sign fml v*›
⟨*proof*⟩

lemma *fml-ext-preserves-sign*: *is-Pos v ≡ is-Pos* (*fml-ext v*)
⟨*proof*⟩

lemma *to-V-fml-ext* [*simp*]: ‹*to-V* (*fml-ext v*) = *fml* (*to-V v*)›
⟨*proof*⟩

lemma *fml-ext-preserves-val*: ‹*to-V v1 = to-V v2 ⟹ to-V* (*fml-ext v1*) = *to-V* (*fml-ext v2*)›
⟨*proof*⟩

definition *sound-consistent* :: *'v total-interpretation ⇒ bool* **where**
‹*sound-consistent J ≡ ¬* (*sound-cons.entails-neg* (*fml-ext* ‘ (*total-strip J*)) {*Pos bot*})›

definition *propositional-model* :: *'v total-interpretation ⇒* (*'f*, *'v*) *AF set ⇒ bool*
(**infix** $\models_p$ *50*) **where**
‹*J* $\models_p$ $\mathcal{N}$ *≡ bot ∉* ((*proj*$_\perp$ $\mathcal{N}$) *proj*$_J$ *J*)›

lemma ‹*J* $\models_p$ {}›
⟨*proof*⟩

The definition below is essentially the same as the one above since term(*proj*$_\perp$ $\mathcal{N}$) *proj*$_J$ *J* is either empty or contains only bot

definition *propositional-model2* :: *'v total-interpretation ⇒* (*'f*, *'v*) *AF set ⇒ bool*
(**infix** $\models_p 2$ *50*) **where**
‹*J* $\models_p 2$ $\mathcal{N}$ *≡* ({} = ((*proj*$_\perp$ $\mathcal{N}$) *proj*$_J$ *J*))›

lemma *subset-model-p2*: ‹$\mathcal{N}' \subseteq \mathcal{N} \Longrightarrow J \models_p 2\ \mathcal{N} \Longrightarrow J \models_p 2\ \mathcal{N}'$›
⟨*proof*⟩

lemma *subset-not-model*: ‹$\neg\ J \models_p 2\ \mathcal{N} \Longrightarrow \mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2 \Longrightarrow J \models_p 2\ \mathcal{N}_1 \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}_2$›
⟨*proof*⟩

lemma *supset-not-model-p2*: ‹$\mathcal{N}' \subseteq \mathcal{N} \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}' \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}$›
⟨*proof*⟩

fun *sign-to-atomic-formula* :: *'v sign ⇒ 'v formula* **where**
‹*sign-to-atomic-formula* (*Pos v*) = *Atom v*› |
‹*sign-to-atomic-formula* (*Neg v*) = *Not* (*Atom v*)›

definition *sign-set-to-formula-set* :: *'v sign set ⇒ 'v formula set* **where**
‹*sign-set-to-formula-set A = sign-to-atomic-formula* ‘ *A*›

lemma *form-shape-sign-set*: ‹∀*f*∈*sign-set-to-formula-set A*. ∃*v*. *f = Atom v ∨ f = Not* (*Atom v*)›

⟨*proof*⟩

definition *AF-to-formula-set* :: (*'f*, *'v*) *AF* ⇒ *'v formula set* **where**

‹*AF-to-formula-set* $\mathcal{C}$ = *sign-set-to-formula-set* (*neg* ‘ *fset* (*A-of* $\mathcal{C}$))›

definition *AF-to-formula* :: (*'f*, *'v*) *AF* ⇒ *'v formula* **where**
‹*AF-to-formula* $\mathcal{C}$ = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›

lemma *form-shape-AF*: ‹∀ *f*∈*AF-to-formula-set* $\mathcal{C}$. ∃ *v*. *f* = *Atom v* ∨ *f* = *Not* (*Atom v*)›
⟨*proof*⟩

definition *AF-proj-to-formula-set-set* :: (*'f*, *'v*) *AF set* ⇒ *'v formula set set* **where**

‹*AF-proj-to-formula-set-set* $\mathcal{N}$ = *AF-to-formula-set* ‘ (*proj*$_\perp$ $\mathcal{N}$)›

definition *AF-proj-to-formula-set* :: (*'f*, *'v*) *AF set* ⇒ *'v formula set* **where**
‹*AF-proj-to-formula-set* $\mathcal{N}$ = *AF-to-formula* ‘ (*proj*$_\perp$ $\mathcal{N}$)›

definition *AF-assertions-to-formula* :: (*'f*, *'v*) *AF* ⇒ *'v formula* **where**
‹*AF-assertions-to-formula* $\mathcal{C}$ = *BigAnd* (*map sign-to-atomic-formula* (*list-of-fset* (*A-of* $\mathcal{C}$)))›

definition *AF-assertions-to-formula-set* :: (*'f*, *'v*) *AF set* ⇒ *'v formula set* **where**
‹*AF-assertions-to-formula-set* $\mathcal{N}$ = *AF-assertions-to-formula* ‘ $\mathcal{N}$›

lemma *F-to-C-set*: ‹∀ *F*∈*AF-proj-to-formula-set-set* $\mathcal{N}$. ∃ $\mathcal{C}$∈*proj*$_\perp$ $\mathcal{N}$. *F* = *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* $\mathcal{C}$)›
⟨*proof*⟩

lemma *F-to-C*: ‹∀ *F*∈*AF-proj-to-formula-set* $\mathcal{N}$. ∃ $\mathcal{C}$∈*proj*$_\perp$ $\mathcal{N}$. *F* = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
⟨*proof*⟩

lemma *C-to-F-set*: ‹∀ $\mathcal{C}$∈*proj*$_\perp$ $\mathcal{N}$. ∃ *F*∈*AF-proj-to-formula-set-set* $\mathcal{N}$. *F* = *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* $\mathcal{C}$)›
⟨*proof*⟩

lemma *C-to-F*: ‹∀ $\mathcal{C}$∈*proj*$_\perp$ $\mathcal{N}$. ∃ *F*∈*AF-proj-to-formula-set* $\mathcal{N}$. *F* = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
⟨*proof*⟩

lemma *form-shape-proj*: ‹∀ *f*∈$\bigcup$ (*AF-proj-to-formula-set-set* $\mathcal{N}$). ∃ *v*. *f* = *Atom v* ∨ *f* = *Not* (*Atom v*)›
⟨*proof*⟩

definition *to-valuation* :: *'v total-interpretation* ⇒ *'v valuation* **where**

‹*to-valuation* $J = (\lambda a.\ Pos\ a \in_t J)$›

lemma *val-strip-pos*: ‹*to-valuation* $J\ a \equiv Pos\ a \in$ *total-strip* J›
⟨*proof*⟩

lemma *val-strip-neg*: ‹$(\neg$ *to-valuation* $J\ a) = (Neg\ a \in$ *total-strip* $J)$›
⟨*proof*⟩

lemma *equiv-prop-entails*: ‹$(J \models_p \mathcal{N}) \longleftrightarrow (J \models_p 2\ \mathcal{N})$›
⟨*proof*⟩

definition *propositional-model3* :: *'v total-interpretation* ⇒ *('f, 'v) AF set* ⇒ *bool*
(**infix** $\models_p 3$ *50*) **where**
‹$J \models_p 3\ \mathcal{N} \equiv (\forall F{\in}$*AF-proj-to-formula-set-set* $\mathcal{N}.\ \exists f{\in}F.$ *formula-semantics* (*to-valuation* J) $f)$›

lemma *equiv-prop-entail2-sema*:
‹$(J \models_p 2\ \mathcal{N}) \longleftrightarrow (J \models_p 3\ \mathcal{N})$›
⟨*proof*⟩

lemma *equiv-prop-entail2-sema2*:
‹$(J \models_p 2\ \mathcal{N}) \longleftrightarrow (\forall F{\in}$*AF-proj-to-formula-set* $\mathcal{N}.$ *formula-semantics* (*to-valuation* J) $F)$›
⟨*proof*⟩

lemma *equiv-prop-entail-sema*:
‹$(J \models_p \mathcal{N}) \longleftrightarrow (J \models_p 3\ \mathcal{N})$›
⟨*proof*⟩

lemma ‹f ‘ *fset* $A =$ *set* (*map* f (*list-of-fset* A))›
⟨*proof*⟩

lemma *equiv-prop-sema1-sema2*:
‹$(J \models_p 3\ \mathcal{N}) \longleftrightarrow$
($\forall F{\in}$*AF-proj-to-formula-set* $\mathcal{N}.$ *formula-semantics* (*to-valuation* J) $F)$›
⟨*proof*⟩

lemma *equiv-enabled-assertions-sema*:
‹(*enabled-set* $\mathcal{N}\ J) \longleftrightarrow (\forall F{\in}$*AF-assertions-to-formula-set* $\mathcal{N}.$ *formula-semantics* (*to-valuation* J) $F)$›
⟨*proof*⟩

definition *sound-propositional-model* :: *'v total-interpretation* ⇒ *('f, 'v) AF set* ⇒ *bool*
(**infix** $\models s_p$ *50*) **where**

‹$J \models s_p\ \mathcal{N} \equiv (bot \notin ((enabled\text{-}projection\ (propositional\text{-}projection\ \mathcal{N})\ J)) \vee$
$\neg\ sound\text{-}consistent\ J)$›

definition *propositionally-unsatisfiable* :: $('f, 'v)\ AF\ set \Rightarrow bool$ **where**
‹*propositionally-unsatisfiable* $\mathcal{N} \equiv \forall J.\ \neg\ (J \models_p \mathcal{N})$›

lemma *unsat-simp*:
assumes
‹$\neg\ sat\ (S' \cup S :: 'v\ formula\ set)$›
‹$sat\ S'$›
‹$\bigcup\ (atoms\ `\ S') \cap \bigcup\ (atoms\ `\ S) = \{\}$›
shows
‹$\neg\ sat\ S$›
⟨*proof*⟩

lemma *proj-to-form-un*: ‹*AF-proj-to-formula-set* $(A \cup B) =$
AF-proj-to-formula-set $A \cup$ *AF-proj-to-formula-set* B›
⟨*proof*⟩

lemma *unsat-AF-simp*:
assumes
‹$\neg\ sat\ (AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ (S' \cup S))$›
‹$sat\ (AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ S')$›
‹$\bigcup\ (atoms\ `\ (AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ S')) \cap \bigcup\ (atoms\ `\ (AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ S)) = \{\}$›
shows
‹$\neg\ sat\ (AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ S)$›
⟨*proof*⟩

lemma *set-list-of-fset*[*simp*]: ‹*set* (*list-of-fset* A) = *fset* A›
⟨*proof*⟩

lemma *vars-in-assertion*: ‹*to-V* ` (*set* (*list-of-fset* $A$)) = *to-V* ` (*fset* A)›
⟨*proof*⟩

lemma *atoms-bigor*: ‹*atoms* (*BigOr* L) = $\bigcup$ (*atoms* ` (*set* L))›
⟨*proof*⟩

lemma *atoms-neg*: ‹*atoms* (*sign-to-atomic-formula* (*neg* A)) = *atoms* (*sign-to-atomic-formula* A)›
⟨*proof*⟩

lemma *set-maps-list-of-fset*: ‹*set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* A))) =
sign-to-atomic-formula ` *neg* ` *fset* A›
⟨*proof*⟩

lemma *atoms-to-V-mono*: ‹*atoms* (*sign-to-atomic-formula* A) = {*to-V* A}›
⟨*proof*⟩

lemma *atoms-to-V*: ‹$\bigcup$(*atoms* ‘ *sign-to-atomic-formula* ‘ *A*) = *to-V* ‘ *A*›
⟨*proof*⟩

lemma *atoms-to-V-AF*: ‹*atoms* (*AF-to-formula* (*Pair C A*)) = *to-V* ‘ (*fset A*)›
⟨*proof*⟩

lemma *atoms-to-V-A-of*: ‹*atoms* (*AF-to-formula* $\mathcal{C}$) = *to-V* ‘ (*fset* (*A-of* $\mathcal{C}$))›
⟨*proof*⟩

lemma *atoms-to-V-un*: ‹$\bigcup$(*atoms* ‘ *AF-to-formula* ‘ $\mathcal{S}$) = $\bigcup${*to-V* ‘ *fset A* |*A*. *A* ∈ *A-of* ‘ $\mathcal{S}$}›
⟨*proof*⟩

lemma *atoms-simp*: ‹$\bigcup$ (*atoms* ‘ (*AF-proj-to-formula-set S*)) = *to-V* ‘ $\bigcup$ (*fset* ‘ (*A-of* ‘ (*proj*$_\perp$ *S*)))›
⟨*proof*⟩

lemma *val-from-interp*: ‹∀ $\mathcal{A}$. ∃ *J*. $\mathcal{A}$ = *to-valuation J*›
⟨*proof*⟩

lemma *interp-from-val*: ‹∀ *J*. ∃ $\mathcal{A}$. $\mathcal{A}$ = *to-valuation J*›
⟨*proof*⟩

lemma *compactness-unsat*: ‹(¬ *sat* (*S*::*'v formula set*)) ⟷ (∃ *s*⊆*S*. *finite s* ∧ ¬ *sat s*)›
⟨*proof*⟩

lemma *never-enabled-finite-subset*:
‹∀ *J*. ¬ *enabled-set* $\mathcal{N}$ *J* ⟹ ∃ $\mathcal{N}'$⊆$\mathcal{N}$. *finite* $\mathcal{N}'$ ∧ (∀ *J*. ¬ *enabled-set* $\mathcal{N}'$ *J*)›
⟨*proof*⟩

lemma *compactness-AF-proj*: ‹(∀ *J*. ¬ *J* ⊨$_p$2 $\mathcal{N}$) ⟷ (∃$\mathcal{N}'$⊆$\mathcal{N}$. *finite* $\mathcal{N}'$ ∧ (∀ *J*. ¬ *J* ⊨$_p$2 $\mathcal{N}'$))›
⟨*proof*⟩

lemma *prop-unsat-compactness*:
‹*propositionally-unsatisfiable A* ⟹ ∃ *B* ⊆ *A*. *finite B* ∧ *propositionally-unsatisfiable B*›
⟨*proof*⟩

definition $\mathcal{E}$*-from* :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **where**
‹$\mathcal{E}$*-from* $\mathcal{N}$ ≡ {*Pair bot* {|*neg a*|} |*a*. ∃$\mathcal{C}$∈$\mathcal{N}$. *a* ∈ *fset* (*A-of* $\mathcal{C}$)}›

lemma *prop-proj-*$\mathcal{E}$*-from*: ‹*proj*$_\perp$ ($\mathcal{E}$*-from* $\mathcal{N}$) = $\mathcal{E}$*-from* $\mathcal{N}$›
⟨*proof*⟩

lemma *prop-proj-sub*: ‹$proj_\perp\ \mathcal{N} = \mathcal{N} \Longrightarrow \mathcal{N}' \subseteq \mathcal{N} \Longrightarrow proj_\perp\ \mathcal{N}' = \mathcal{N}'$›
⟨*proof*⟩

lemma *prop-proj-distrib*: ‹$proj_\perp\ (A \cup B) = proj_\perp\ A \cup proj_\perp\ B$›
⟨*proof*⟩

lemma *v-in-*$\mathcal{E}$: ‹*Pair bot* $\{|Pos\ v|\} \in \mathcal{E}$*-from* $\mathcal{N} \vee$ *Pair bot* $\{|Neg\ v|\} \in \mathcal{E}$*-from* $\mathcal{N}$
$\Longrightarrow$
$\exists \mathcal{C} \in \mathcal{N}.\ v \in$ *to-V* ‘ (*fset* (*A-of* $\mathcal{C}$))›
⟨*proof*⟩

lemma *a-in-*$\mathcal{E}$: ‹$\exists J.\ J \models_p 2\ \mathcal{E}$*-from* $\mathcal{N} \Longrightarrow$ *Pair bot* $\{|neg\ a|\} \in \mathcal{E}$*-from* $\mathcal{N} \Longrightarrow$
$\neg$ (*Pair bot* $\{|a|\} \in \mathcal{E}$*-from* $\mathcal{N}$)›
⟨*proof*⟩

lemma *equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$:
shows ‹$J \models_p 2\ \mathcal{E}$*-from* $\mathcal{N} \longleftrightarrow$ *enabled-set* $\mathcal{N}\ J$›
⟨*proof*⟩

definition *AF-entails* :: $('f, 'v)\ AF\ set \Rightarrow ('f, 'v)\ AF\ set \Rightarrow bool$ (**infix** $\models_{AF}$ *50*)
where
‹*AF-entails* $\mathcal{M}\ \mathcal{N} \equiv (\forall J.$ (*enabled-set* $\mathcal{N}\ J \longrightarrow \mathcal{M}\ proj_J\ J \models$ *F-of* ‘ $\mathcal{N}$))›

lemma *prop-unsat-to-AF-entails-bot*: ‹*propositionally-unsatisfiable* $\mathcal{M} \Longrightarrow \mathcal{M} \models_{AF}$
{*to-AF bot*}›
⟨*proof*⟩

lemma ‹*enabled-set* $\{\}\ J$›
⟨*proof*⟩

lemma ‹$(\forall J. \neg$ (*enabled-set* $\mathcal{N}\ J)) \Longrightarrow (\mathcal{M} \models_{AF} \mathcal{N})$›
⟨*proof*⟩

definition *AF-entails-sound* :: $('f, 'v)\ AF\ set \Rightarrow ('f, 'v)\ AF\ set \Rightarrow bool$ (**infix**
$\models s_{AF}$ *50*) **where**
‹*AF-entails-sound* $\mathcal{M}\ \mathcal{N} \equiv (\forall J.$ (*enabled-set* $\mathcal{N}\ J \longrightarrow$
sound-cons.entails-neg ((*fml-ext* ‘ (*total-strip* J)) $\cup$ (*Pos* ‘ ($\mathcal{M}\ proj_J\ J$))) (*Pos* ‘
F-of ‘ $\mathcal{N}$)))›

lemma *distrib-proj*: ‹$\mathcal{M} \cup \mathcal{N}\ proj_J\ J = (\mathcal{M}\ proj_J\ J) \cup (\mathcal{N}\ proj_J\ J)$›
⟨*proof*⟩

lemma *distrib-proj-singleton*: ‹$\mathcal{M} \cup \{\mathcal{C}\}\ proj_J\ J = (\mathcal{M}\ proj_J\ J) \cup (\{\mathcal{C}\}\ proj_J\ J)$›
⟨*proof*⟩

lemma *enabled-union2*: ‹*enabled-set* $(\mathcal{M} \cup \mathcal{N})\ J \Longrightarrow$ *enabled-set* $\mathcal{N}\ J$›
⟨*proof*⟩

lemma *enabled-union1*: ‹*enabled-set* $(\mathcal{M} \cup \mathcal{N})\ J \Longrightarrow$ *enabled-set* $\mathcal{M}\ J$›

⟨*proof*⟩

lemma *finite-subset-image-strong*:
assumes *finite U* **and**
(∀ *C* ∈ *U*. (∃ *D* ∈ *W*. *P D* = *C* ∧ *Q D*))
shows ∃ *W'*⊆*W*. *finite W'* ∧ *U* = *P* ‘ *W'* ∧ (∀ *D'*∈ *W'*. *Q D'*)
⟨*proof*⟩

lemma *all-to-ex*: ‹∀ *x*. *P x* ⟹ ∃ *x*. *P x*› **for** *P* ⟨*proof*⟩

lemma *three-skolems*:
assumes ‹⋀*U*. *P U* ⟹ ∃ *X Y Z*. *Q U X Y Z*›
shows ‹(⋀*X-of Y-of Z-of*. (⋀*U*. *P U* ⟹ *Q U* (*X-of U*) (*Y-of U*) (*Z-of U*))
⟹ *thesis*) ⟹ *thesis*›
⟨*proof*⟩

lemma *finite-subset-with-prop*:
assumes ‹∃ *Js*. *A* = *f* ‘ *Js* ∧ (∀ *J*∈*Js*. *P J*)› **and**
‹*finite C*› **and**
‹*B* = *C* ∩ *A*›
shows ‹∃ *Js*. *B* = *f* ‘ *Js* ∧ (∀ *J*∈*Js*. *P J*) ∧ *finite Js*›
⟨*proof*⟩

lemma *to-V-neg* [*simp*]: ‹*to-V* (*neg a*) = *to-V a*›
⟨*proof*⟩

sublocale *AF-cons-rel*: *consequence-relation to-AF bot AF-entails*
⟨*proof*⟩

sublocale *neg-ext-sound-cons-rel*: *consequence-relation Pos bot sound-cons.entails-neg*
⟨*proof*⟩

lemma *AF-ext-sound-cons-rel*: ‹*consequence-relation* (*to-AF bot*) *AF-entails-sound*›
⟨*proof*⟩

interpretation *AF-sound-cons-rel*: *consequence-relation to-AF bot AF-entails-sound*
⟨*proof*⟩

lemma *f-of-to-AF* [*simp*]: ‹*F-of* ‘ *to-AF* ‘ *N* = *N*›
⟨*proof*⟩

lemma *to-AF-proj-J* [*simp*]: ‹*to-AF* ‘ *M proj*$_J$ *J* = *M*›
⟨*proof*⟩

lemma *enabled-to-AF-set* [*simp*]: ‹*enabled-set* (*to-AF* ‘ *N*) *J*›
⟨*proof*⟩

lemma *pos-not-pos-empty* [*simp*]: ‹{*to-V C* |*C*. *C* ∈ *Pos* ‘ *N* ∧ ¬ *is-Pos C*} = {}›
⟨*proof*⟩

lemma *pos-not-pos-simp* [*simp*]:
‹{*to-V C* |*C*. *C* ∈ *U* ∪ *Pos* ‘ *M* ∧ ¬ *is-Pos C*} = {*to-V C* |*C*. *C* ∈ *U* ∧ ¬ *is-Pos C*}›
⟨*proof*⟩

lemma *pos-pos-simp* [*simp*]: ‹{*to-V C* |*C*. *C* ∈ *Pos* ‘ *F-of* ‘ *N* ∧ *is-Pos C*} = *F-of* ‘ *N*›
⟨*proof*⟩

lemma *proj-F-of* [*simp*]: ‹{*C*. *F-of C* ∈ *M*} *proj*$_J$ *J* = *M*›
⟨*proof*⟩

lemma *f-of-F-of* [*simp*]: ‹*F-of* ‘ {*C*. *F-of C* ∈ *M*} = *M*›
⟨*proof*⟩

lemma *set-on-union-triple-split*: ‹{*f C* |*C*. *C* ∈ *M* ∪ *N* ∪ *g J* ∧ *l C J*} = {*f C* |*C*. *C* ∈ *M* ∧ *l C J*} ∪
{*f C* |*C*. *C* ∈ *N* ∧ *l C J*} ∪ {*f C* |*C*. *C* ∈ *g J* ∧ *l C J*}›
⟨*proof*⟩

lemma *not-enabled-enabled-empty* [*simp*]:
‹{*F-of C* |*C*. *C* ∈ {*C*. *F-of C* ∈ *Q*′ ∧ ¬ *enabled C J*} ∧ *enabled C J*} = {}›
⟨*proof*⟩

lemma *f-of-simp-enabled* [*simp*]: ‹{*F-of C* |*C*. *F-of C* ∈ *M* ∧ *enabled C J*} = *M*›
⟨*proof*⟩

lemma *f-of-enabled-simp* [*simp*]: ‹*F-of* ‘ {*C*. *F-of C* ∈ *M* ∧ *enabled C J*} = *M*›
⟨*proof*⟩

lemma ‹(*to-AF* ‘ *M* ⊨$_{AF}$ *to-AF* ‘ *N*) ≡ (*M* ⊨ *N*)›
⟨*proof*⟩

sublocale *ext-cons-rel-std*: *consequence-relation Pos* (*to-AF bot*) *AF-cons-rel.entails-neg*
⟨*proof*⟩

sublocale *sound-cons-rel*: *consequence-relation Pos bot sound-cons.entails-neg*
⟨*proof*⟩

lemma *pos-in-pos-simp* [*simp*]: ‹{*C*. *Pos C* ∈ *Pos* ‘ *N*} = *N*› ⟨*proof*⟩
lemma *neg-not-in-pos-simp* [*simp*]: ‹{*C*. *Neg C* ∈ *Pos* ‘ *N*} = {}› ⟨*proof*⟩
lemma *neg-in-pos-simp* [*simp*]: ‹{*C*. *Neg C* ∈ *P* ∨ *Neg C* ∈ *Pos* ‘ *M*} = {*C*. *Neg C* ∈ *P*}› ⟨*proof*⟩
lemma *pos-in-pos-partial-simp* [*simp*]: ‹{*C*. *Pos C* ∈ *P* ∨ *Pos C* ∈ *Pos* ‘ *M*} = {*C*. *Pos C* ∈ *P*} ∪ *M*› ⟨*proof*⟩

lemma ‹(*to-AF* ‘ $M \models s_{AF}$ *to-AF* ‘ N) ≡ ($M \models s$ N)›
⟨*proof*⟩

lemma *strong-entails-bot-cases*: ‹$\mathcal{N} \cup \{AF.Pair\ bot\ A\} \models s_{AF}\ \{AF.Pair\ bot\ B\}$ ⟹
(∀ J. *fset* $B \subseteq$ *total-strip* $J \longrightarrow$
(*fml-ext* ‘ *total-strip* $J \cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J)) $\models s_\sim$ {*Pos bot*} ∨ *fset* $A \subseteq$ *total-strip* J)›
⟨*proof*⟩

lemma *strong-entails-bot-cases-Union*:
‹$\mathcal{N} \cup \mathcal{M} \models s_{AF}\ \{AF.Pair\ bot\ B\}$ ⟹ (∀ $x \in \mathcal{M}$. *F-of* x = *bot*) ⟹
(∀ J. *fset* $B \subseteq$ *total-strip* $J \longrightarrow$
(*fml-ext* ‘ *total-strip* $J \cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J)) $\models s_\sim$ {*Pos bot*} ∨
(∃ $A \in$ *A-of* ‘ $\mathcal{M}$. *fset* $A \subseteq$ *total-strip* J))›
⟨*proof*⟩

lemma *AF-entails-sound-right-disjunctive*: ‹(∃ $\mathcal{C}' \in A$. $\mathcal{M} \models s_{AF}\ \{\mathcal{C}'\}$) ⟹ $\mathcal{M}$ $\models s_{AF}$ A›
⟨*proof*⟩

6.1 Local saturation

To fully capture completeness for splitting, we need to use weaker notions of saturation and fairness.

definition *locally-saturated* :: ‹('*f*, '*v*) *AF set* ⇒ *bool*› **where**
‹*locally-saturated* $\mathcal{N}$ ≡
to-AF bot ∈ $\mathcal{N}$ ∨
(∃ J :: '*v total-interpretation*. $J \models_p \mathcal{N}$ ∧ *saturated* ($\mathcal{N}$ $proj_J$ J))›

definition *locally-fair* :: ‹('*f*, '*v*) *AF set infinite-llist* ⇒ *bool*› **where**
‹*locally-fair* $\mathcal{N}i$ ≡
(∃ i. *to-AF bot* ∈ *llnth* $\mathcal{N}i$ i)
∨ (∃ J :: '*v total-interpretation*. $J \models_p$ *lim-inf* $\mathcal{N}i$ ∧
Inf-from (*lim-inf* $\mathcal{N}i$ $proj_J$ J) ⊆ (⋃ i. Red_I (*llnth* $\mathcal{N}i$ i $proj_J$ J)))›

end

locale *strong-statically-complete-annotated-calculus* =
calculus-with-annotated-consrel bot Inf entails entails-sound Red_I Red_F *fml asn*
+ *S-calculus*: *calculus to-AF bot SInf AF-entails* $SRed_I$ $SRed_F$
for
bot :: '*f* **and**
Inf :: ‹'*f inference set*› **and**

entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*) **and**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set* **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*› **and**
SInf :: (*'f*, *'v*) *AF inference set* **and**
$SRed_I$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
$SRed_F$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*
+ **assumes**
strong-static-completeness: ‹*locally-saturated* $\mathcal{N}$ ⟹ $\mathcal{N} \models_{AF} \{to\text{-}AF\ bot\}$ ⟹ *to-AF bot* ∈ $\mathcal{N}$›

locale *strong-dynamically-complete-annotated-calculus* =
calculus-with-annotated-consrel bot Inf entails entails-sound Red_I Red_F *fml asn*
+ *S-calculus*: *calculus to-AF bot SInf AF-entails* $SRed_I$ $SRed_F$
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*) **and**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set* **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*› **and**
SInf :: (*'f*, *'v*) *AF inference set* **and**
$SRed_I$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
$SRed_F$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*
+ **assumes**
strong-dynamic-completeness: ‹*is-derivation S-calculus.derive* $\mathcal{N}$ *i* ⟹ *locally-fair* $\mathcal{N}$ *i* ⟹
llhd $\mathcal{N}$ *i* $\models_{AF} \{to\text{-}AF\ bot\}$ ⟹ ∃ *i*. *to-AF bot* ∈ *llnth* $\mathcal{N}$ *i i*›

end

7 Lifting to Non-ground Calculi

The section 3.1 to 3.3 of the report are covered by the current section. Various forms of lifting are proven correct. These allow to obtain the dynamic refutational completeness of a non-ground calculus from the static refutational completeness of its ground counterpart.

theory *Light-Lifting-to-Non-Ground-Calculi*
imports
Saturation-Framework.Intersection-Calculus
Saturation-Framework.Calculus-Variations
Well-Quasi-Orders.Minimal-Elements
begin

7.1 Standard Lifting

locale *light-standard-lifting = inference-system Inf-F +*
ground: calculus Bot-G Inf-G entails-G Red-I-G Red-F-G
for
Inf-F :: ‹'f inference set› **and**
Bot-G :: ‹'g set› **and**
Inf-G :: ‹'g inference set› **and**
entails-G :: ‹'g set ⇒ 'g set ⇒ bool› (**infix** $\models G$ *50*) **and**
Red-I-G :: ‹'g set ⇒ 'g inference set› **and**
Red-F-G :: ‹'g set ⇒ 'g set›
+ fixes
Bot-F :: ‹'f set› **and**
$\mathcal{G}$*-F :: ‹'f ⇒ 'g set›* **and**
$\mathcal{G}$*-I :: ‹'f inference ⇒ 'g inference set option›*
assumes
Bot-F-not-empty: Bot-F ≠ {} **and**
Bot-map-not-empty: ‹B ∈ Bot-F ⟹ $\mathcal{G}$*-F B ≠ {}›* **and**
Bot-map: ‹B ∈ Bot-F ⟹ $\mathcal{G}$*-F B ⊆ Bot-G›* **and**

inf-map: ‹ι ∈ Inf-F ⟹ $\mathcal{G}$*-I ι ≠ None ⟹ the (*$\mathcal{G}$*-I ι) ⊆ Red-I-G (*$\mathcal{G}$*-F (concl-of ι))›*
begin

abbreviation $\mathcal{G}$*-Fset :: ‹'f set ⇒ 'g set›* **where**
‹$\mathcal{G}$*-Fset N ≡ ⋃ (*$\mathcal{G}$*-F ‘ N)›*

lemma $\mathcal{G}$*-subset: ‹N1 ⊆ N2 ⟹* $\mathcal{G}$*-Fset N1 ⊆* $\mathcal{G}$*-Fset N2›* ⟨*proof*⟩

abbreviation *entails-*$\mathcal{G}$ *:: ‹'f set ⇒ 'f set ⇒ bool›* (**infix** $\models\mathcal{G}$ *50*) **where**
‹N1 ⊨$\mathcal{G}$ *N2 ≡* $\mathcal{G}$*-Fset N1 ⊨G* $\mathcal{G}$*-Fset N2›*

lemma *subs-Bot-G-entails*:
assumes
not-empty: ‹sB ≠ {}› **and**
in-bot: ‹sB ⊆ Bot-G›
shows *‹sB ⊨G N›*
⟨*proof*⟩

sublocale *consequence-relation Bot-F entails-*$\mathcal{G}$
⟨*proof*⟩

definition *Red-I-*$\mathcal{G}$ *:: 'f set ⇒ 'f inference set* **where**
‹Red-I-$\mathcal{G}$ *N = {ι ∈ Inf-F. (*$\mathcal{G}$*-I ι ≠ None ∧ the (*$\mathcal{G}$*-I ι) ⊆ Red-I-G (*$\mathcal{G}$*-Fset N))*
∨ ($\mathcal{G}$*-I ι = None ∧* $\mathcal{G}$*-F (concl-of ι) ⊆* $\mathcal{G}$*-Fset N ∪ Red-F-G (*$\mathcal{G}$*-Fset N))}›*

definition *Red-F-*$\mathcal{G}$ *:: 'f set ⇒ 'f set* **where**
‹Red-F-$\mathcal{G}$ *N = {C. ∀ D ∈* $\mathcal{G}$*-F C. D ∈ Red-F-G (*$\mathcal{G}$*-Fset N)}›*
end

7.2 Strong Standard Lifting

locale *light-strong-standard-lifting* = *inference-system Inf-F* +
ground: *calculus Bot-G Inf-G entails-G Red-I-G Red-F-G*
for
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
Inf-G :: ‹*'g inference set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** ⊨*G* 50) **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**
Red-F-G :: ‹*'g set* ⇒ *'g set*›
+ **fixes**
Bot-F :: ‹*'f set*› **and**
$\mathcal{G}$*-F* :: ‹*'f* ⇒ *'g set*› **and**
$\mathcal{G}$*-I* :: ‹*'f inference* ⇒ *'g inference set option*›
assumes
Bot-F-not-empty: *Bot-F* ≠ {} **and**
Bot-map-not-empty: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ≠ {}› **and**
Bot-map: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ⊆ *Bot-G*› **and**

strong-inf-map: ‹*ι* ∈ *Inf-F* ⟹ $\mathcal{G}$*-I ι* ≠ *None* ⟹ *concl-of* ‘ (*the* ($\mathcal{G}$*-I ι*)) ⊆ ($\mathcal{G}$*-F* (*concl-of ι*))› **and**
inf-map-in-Inf: ‹*ι* ∈ *Inf-F* ⟹ $\mathcal{G}$*-I ι* ≠ *None* ⟹ *the* ($\mathcal{G}$*-I ι*) ⊆ *Inf-G*›
begin

sublocale *light-standard-lifting Inf-F Bot-G Inf-G* (⊨*G*) *Red-I-G Red-F-G Bot-F* $\mathcal{G}$*-F* $\mathcal{G}$*-I*
⟨*proof*⟩

end

7.3 Lifting with a Family of Tiebreaker Orderings

locale *light-tiebreaker-lifting* =
empty-ord?: *light-standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G Red-F-G Bot-F* $\mathcal{G}$*-F* $\mathcal{G}$*-I*
for
Bot-F :: ‹*'f set*› **and**
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** ⊨*G* 50) **and**
Inf-G :: ‹*'g inference set*› **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**
Red-F-G :: ‹*'g set* ⇒ *'g set*› **and**
$\mathcal{G}$*-F* :: *'f* ⇒ *'g set* **and**
$\mathcal{G}$*-I* :: *'f inference* ⇒ *'g inference set option*
+ **fixes**
Prec-F-g :: ‹*'g* ⇒ *'f* ⇒ *'f* ⇒ *bool*›
assumes
all-wf: *minimal-element* (*Prec-F-g g*) *UNIV*

begin

definition *Red-F-$\mathcal{G}$* :: *'f set* ⇒ *'f set* **where**
‹*Red-F-$\mathcal{G}$ N* = {*C*. ∀ *D* ∈ *$\mathcal{G}$-F C*. *D* ∈ *Red-F-G* (*$\mathcal{G}$-Fset N*) ∨ (∃ *E* ∈ *N*. *Prec-F-g D E C* ∧ *D* ∈ *$\mathcal{G}$-F E*)}›

lemma *Prec-trans*:
assumes
‹*Prec-F-g D A B*› **and**
‹*Prec-F-g D B C*›
shows
‹*Prec-F-g D A C*›
⟨*proof*⟩

lemma *prop-nested-in-set*: *D* ∈ *P C* ⟹ *C* ∈ {*C*. ∀ *D* ∈ *P C*. *A D* ∨ *B C D*} ⟹ *A D* ∨ *B C D*
⟨*proof*⟩

lemma *Red-F-$\mathcal{G}$-equiv-def*:
‹*Red-F-$\mathcal{G}$ N* = {*C*. ∀ *Di* ∈ *$\mathcal{G}$-F C*. *Di* ∈ *Red-F-G* (*$\mathcal{G}$-Fset N*) ∨
(∃ *E* ∈ (*N* − *Red-F-$\mathcal{G}$ N*). *Prec-F-g Di E C* ∧ *Di* ∈ *$\mathcal{G}$-F E*)}›
⟨*proof*⟩

lemma *not-red-map-in-map-not-red*: ‹*$\mathcal{G}$-Fset N* − *Red-F-G* (*$\mathcal{G}$-Fset N*) ⊆ *$\mathcal{G}$-Fset* (*N* − *Red-F-$\mathcal{G}$ N*)›
⟨*proof*⟩

lemma *Red-F-Bot-F*: ‹*B* ∈ *Bot-F* ⟹ *N* ⊨*$\mathcal{G}$* {*B*} ⟹ *N* − *Red-F-$\mathcal{G}$ N* ⊨*$\mathcal{G}$* {*B*}›
⟨*proof*⟩

lemma *Red-F-of-subset-F*: ‹*N* ⊆ *N'* ⟹ *Red-F-$\mathcal{G}$ N* ⊆ *Red-F-$\mathcal{G}$ N'*›
⟨*proof*⟩

lemma *Red-I-of-subset-F*: ‹*N* ⊆ *N'* ⟹ *Red-I-$\mathcal{G}$ N* ⊆ *Red-I-$\mathcal{G}$ N'*›
⟨*proof*⟩

lemma *Red-F-of-Red-F-subset-F*: ‹*N'* ⊆ *Red-F-$\mathcal{G}$ N* ⟹ *Red-F-$\mathcal{G}$ N* ⊆ *Red-F-$\mathcal{G}$* (*N* − *N'*)›
⟨*proof*⟩

lemma *Red-I-of-Red-F-subset-F*: ‹*N'* ⊆ *Red-F-$\mathcal{G}$ N* ⟹ *Red-I-$\mathcal{G}$ N* ⊆ *Red-I-$\mathcal{G}$* (*N* − *N'*) ›

⟨*proof*⟩

lemma *Red-I-of-Inf-to-N-F*:
assumes
i-in: ‹$\iota \in$ *Inf-F*› **and**
concl-i-in: ‹*concl-of* $\iota \in N$›
shows
‹$\iota \in$ *Red-I-*$\mathcal{G}$ N ›
⟨*proof*⟩

sublocale *calculus Bot-F Inf-F entails-*$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
⟨*proof*⟩

end

lemma *wf-empty-rel*: *minimal-element* (λ- -. *False*) *UNIV*
⟨*proof*⟩

lemma *light-standard-empty-tiebreaker-equiv*: *light-standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G*
Red-F-G Bot-F $\mathcal{G}$*-F* $\mathcal{G}$*-I* = *light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G*
Red-F-G $\mathcal{G}$*-F* $\mathcal{G}$*-I* ($\lambda g\ C\ C'$. *False*)
⟨*proof*⟩

context *light-standard-lifting*
begin

interpretation *empt-ord*: *light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G*
Red-F-G $\mathcal{G}$*-F* $\mathcal{G}$*-I* $\lambda g\ C\ C'$. *False*
⟨*proof*⟩

lemma *red-f-equiv*: *empt-ord.Red-F-*$\mathcal{G}$ = *Red-F-*$\mathcal{G}$
⟨*proof*⟩

sublocale *calc?*: *calculus Bot-F Inf-F entails-*$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
⟨*proof*⟩

lemma *grounded-inf-in-ground-inf*: $\iota \in$ *Inf-F* $\Longrightarrow$ $\mathcal{G}$*-I* $\iota \neq$ *None* $\Longrightarrow$ *the* ($\mathcal{G}$*-I* ι) $\subseteq$ *Inf-G*
⟨*proof*⟩

abbreviation *ground-Inf-overapproximated* :: *'f set* $\Rightarrow$ *bool* **where**
ground-Inf-overapproximated $N \equiv$ *ground.Inf-from* ($\mathcal{G}$*-Fset* N)
$\subseteq \{\iota.\ \exists \iota' \in$ *Inf-from* $N.$ $\mathcal{G}$*-I* $\iota' \neq$ *None* $\wedge\ \iota \in$ *the* ($\mathcal{G}$*-I* ι')$\} \cup$ *Red-I-G* ($\mathcal{G}$*-Fset* N)

lemma *sat-inf-imp-ground-red*:
assumes
saturated N **and**
$\iota' \in$ *Inf-from N* **and**
$\mathcal{G}$*-I* $\iota' \neq$ *None* $\wedge$ $\iota \in$ *the* ($\mathcal{G}$*-I* ι')
shows $\iota \in$ *Red-I-G* ($\mathcal{G}$*-Fset N*)
⟨*proof*⟩

lemma *sat-imp-ground-sat*:
saturated N $\Longrightarrow$ *ground-Inf-overapproximated N* $\Longrightarrow$ *ground.saturated* ($\mathcal{G}$*-Fset N*)
⟨*proof*⟩

end

context *light-tiebreaker-lifting*
begin

lemma *saturated-empty-order-equiv-saturated*:
saturated N = *calc.saturated N*
⟨*proof*⟩

lemma *static-empty-order-equiv-static*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$ =
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *empty-ord.Red-F-*$\mathcal{G}$
⟨*proof*⟩

theorem *static-to-dynamic*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *empty-ord.Red-F-*$\mathcal{G}$ =
dynamically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
⟨*proof*⟩

end

7.4 Lifting with a Family of Redundancy Criteria

locale *light-lifting-intersection* = *inference-system Inf-F* +
ground: *inference-system-family Q Inf-G-q* +
ground: *consequence-relation-family Bot-G Q entails-q*

for
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
Inf-G-q :: ‹*'q* ⇒ *'g inference set*› **and**
entails-q :: *'q* ⇒ *'g set* ⇒ *'g set* ⇒ *bool* **and**
Red-I-q :: *'q* ⇒ *'g set* ⇒ *'g inference set* **and**
Red-F-q :: *'q* ⇒ *'g set* ⇒ *'g set*
+ **fixes**
Bot-F :: *'f set* **and**
$\mathcal{G}$*-F-q* :: *'q* ⇒ *'f* ⇒ *'g set* **and**
$\mathcal{G}$*-I-q* :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**
Prec-F-g :: *'g* ⇒ *'f* ⇒ *'f* ⇒ *bool*
assumes
light-standard-lifting-family:
∀ *q* ∈ *Q*. *light-tiebreaker-lifting Bot-F Inf-F Bot-G* (*entails-q q*) (*Inf-G-q q*) (*Red-I-q q*)
(*Red-F-q q*) ($\mathcal{G}$*-F-q q*) ($\mathcal{G}$*-I-q q*) *Prec-F-g*
begin

abbreviation $\mathcal{G}$*-Fset-q* :: *'q* ⇒ *'f set* ⇒ *'g set* **where**
$\mathcal{G}$*-Fset-q q N* ≡ ⋃ ($\mathcal{G}$*-F-q q* ‘ *N*)

definition *Red-I-*$\mathcal{G}$*-q* :: *'q* ⇒ *'f set* ⇒ *'f inference set* **where**
Red-I-$\mathcal{G}$*-q q N* = {ι ∈ *Inf-F*. ($\mathcal{G}$*-I-q q* ι ≠ *None* ∧ *the* ($\mathcal{G}$*-I-q q* ι) ⊆ *Red-I-q q* ($\mathcal{G}$*-Fset-q q N*))
∨ ($\mathcal{G}$*-I-q q* ι = *None* ∧ $\mathcal{G}$*-F-q q* (*concl-of* ι) ⊆ ($\mathcal{G}$*-Fset-q q N* ∪ *Red-F-q q* ($\mathcal{G}$*-Fset-q q N*)))}

definition *Red-F-*$\mathcal{G}$*-empty-q* :: *'q* ⇒ *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$*-empty-q q N* = {*C*. ∀ *D* ∈ $\mathcal{G}$*-F-q q C*. *D* ∈ *Red-F-q q* ($\mathcal{G}$*-Fset-q q N*)}

definition *Red-F-*$\mathcal{G}$*-q* :: *'q* ⇒ *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$*-q q N* =
{*C*. ∀ *D* ∈ $\mathcal{G}$*-F-q q C*. *D* ∈ *Red-F-q q* ($\mathcal{G}$*-Fset-q q N*) ∨ (∃ *E* ∈ *N*. *Prec-F-g D E C* ∧ *D* ∈ $\mathcal{G}$*-F-q q E*)}

abbreviation *entails-*$\mathcal{G}$*-q* :: *'q* ⇒ *'f set* ⇒ *'f set* ⇒ *bool* **where**
entails-$\mathcal{G}$*-q q N1 N2* ≡ *entails-q q* ($\mathcal{G}$*-Fset-q q N1*) ($\mathcal{G}$*-Fset-q q N2*)

lemma *red-crit-lifting-family*:
assumes *q-in*: *q* ∈ *Q*
shows *calculus Bot-F Inf-F* (*entails-*$\mathcal{G}$*-q q*) (*Red-I-*$\mathcal{G}$*-q q*) (*Red-F-*$\mathcal{G}$*-q q*)
⟨*proof*⟩

lemma *red-crit-lifting-family-empty-ord*:
assumes *q-in*: *q* ∈ *Q*
shows *calculus Bot-F Inf-F* (*entails-*$\mathcal{G}$*-q q*) (*Red-I-*$\mathcal{G}$*-q q*) (*Red-F-*$\mathcal{G}$*-empty-q q*)
⟨*proof*⟩

sublocale *consequence-relation-family Bot-F Q entails-$\mathcal{G}$-q*
⟨*proof*⟩

sublocale *intersection-calculus Bot-F Inf-F Q entails-$\mathcal{G}$-q Red-I-$\mathcal{G}$-q Red-F-$\mathcal{G}$-q*
⟨*proof*⟩

abbreviation *entails-$\mathcal{G}$* :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models\cap\mathcal{G}$ *50*) **where**
$(\models\cap\mathcal{G}) \equiv$ *entails*

abbreviation *Red-I-$\mathcal{G}$* :: *'f set* ⇒ *'f inference set* **where**
Red-I-$\mathcal{G}$ ≡ *Red-I*

abbreviation *Red-F-$\mathcal{G}$* :: *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$ ≡ *Red-F*

lemmas *entails-$\mathcal{G}$-def* = *entails-def*
lemmas *Red-I-$\mathcal{G}$-def* = *Red-I-def*
lemmas *Red-F-$\mathcal{G}$-def* = *Red-F-def*

sublocale *empty-ord*: *intersection-calculus Bot-F Inf-F Q entails-$\mathcal{G}$-q Red-I-$\mathcal{G}$-q Red-F-$\mathcal{G}$-empty-q*
⟨*proof*⟩

abbreviation *Red-F-$\mathcal{G}$-empty* :: *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$-empty ≡ *empty-ord.Red-F*

lemmas *Red-F-$\mathcal{G}$-empty-def* = *empty-ord.Red-F-def*

lemma *sat-inf-imp-ground-red-fam-inter*:
assumes
sat-n: *saturated N* **and**
i'-in: *ι'* ∈ *Inf-from N* **and**
q-in: *q* ∈ *Q* **and**
grounding: *$\mathcal{G}$-I-q q ι'* ≠ *None* ∧ *ι* ∈ *the* (*$\mathcal{G}$-I-q q ι'*)
shows *ι* ∈ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)
⟨*proof*⟩

abbreviation *ground-Inf-overapproximated* :: *'q* ⇒ *'f set* ⇒ *bool* **where**
ground-Inf-overapproximated q N ≡
ground.Inf-from-q q (*$\mathcal{G}$-Fset-q q N*)
⊆ {*ι*. ∃*ι'*∈ *Inf-from N*. *$\mathcal{G}$-I-q q ι'* ≠ *None* ∧ *ι* ∈ *the* (*$\mathcal{G}$-I-q q ι'*)} ∪ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)

abbreviation *ground-saturated* :: *'q* ⇒ *'f set* ⇒ *bool* **where**
ground-saturated q N ≡ *ground.Inf-from-q q* (*$\mathcal{G}$-Fset-q q N*) ⊆ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)

lemma *sat-imp-ground-sat-fam-inter*:

saturated N $\Longrightarrow$ *q* $\in$ *Q* $\Longrightarrow$ *ground-Inf-overapproximated q N* $\Longrightarrow$ *ground-saturated q N*
⟨*proof*⟩

lemma *sat-eq-sat-empty-order*: *saturated N* = *empty-ord.saturated N*
⟨*proof*⟩

lemma *static-empty-ord-inter-equiv-static-inter*:
statically-complete-calculus Bot-F Inf-F entails Red-I Red-F =
statically-complete-calculus Bot-F Inf-F entails Red-I Red-F-$\mathcal{G}$*-empty*
⟨*proof*⟩

theorem *stat-eq-dyn-ref-comp-fam-inter*: *statically-complete-calculus Bot-F Inf-F entails Red-I Red-F-*$\mathcal{G}$*-empty* =
dynamically-complete-calculus Bot-F Inf-F entails Red-I Red-F
⟨*proof*⟩

end

end
theory *List-Extra*
imports *Main HOL−Library.Quotient-List*
begin

This theory contains some extra lemmas that were useful in proving some lemmas in `Modular_Splitting_Calculus.thy` and `Lightweight_Avatar.thy`.

lemma *map2-first-is-first* [*simp*]: ‹*length x* = *length y* $\Longrightarrow$ *map2* (λ *x y*. *x*) *x y* = *x*›
⟨*proof*⟩

lemma *map2-second-is-second* [*simp*]: ‹*length A* = *length B* $\Longrightarrow$ *map2* (λ *x y*. *y*) *A B* = *B*›
⟨*proof*⟩

lemma *list-all-exists-is-exists-list-all2*:
assumes ‹*list-all* (λ *x*. ∃ *y*. *P x y*) *xs*›
shows ‹∃ *ys*. *list-all2 P xs ys*›
⟨*proof*⟩

lemma *ball-set-f-to-ball-set-map*: ‹(∀ *x* ∈ *set A*. *P* (*f x*)) ⟷ (∀ *x* ∈ *set* (*map f A*). *P x*)›
⟨*proof*⟩

lemma *list-all-ex-to-ex-list-all2*:
‹*list-all* (λ *x*. ∃ *y*. *P x y*) *A* ⟷ (∃ *ys*. *length A* = *length ys* ∧ *list-all2* (λ *x y*. *P x y*) *A ys*)›
⟨*proof*⟩

lemma *list-all2-to-map*:
assumes *lengths-eq*: ‹*length A* = *length B*›
shows ‹*list-all2* (λ *x y*. *P* (*f x y*)) *A B* ⟷ *list-all P* (*map2 f A B*)›
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩
end

theory *Modular-Splitting-Calculus*
imports
Calculi-And-Annotations
Light-Lifting-to-Non-Ground-Calculi
List-Extra
FSet-Extra
begin

8 Core splitting calculus

In this section, we formalize an abstract version of a splitting calculus. We start by considering only two basic rules:

- Base performs an inference from our inference system;
- Unsat replaces a set of prosopositionally unsatisfiable formulas with ⊥.

locale *annotated-calculus* = *calculus-with-annotated-consrel bot FInf entails entails-sound* $FRed_I$
$FRed_F$ *fml asn*
for *bot* :: *'f* **and**
FInf :: ‹*'f inference set*› **and**
entails :: ‹[*'f set*, *'f set*] ⇒ *bool*› (**infix** ‹⊨› *50*) **and**
entails-sound :: ‹[*'f set*, *'f set*] ⇒ *bool*› (**infix** ‹⊨*s*› *50*) **and**
$FRed_I$ :: ‹*'f set* ⇒ *'f inference set*› **and**
$FRed_F$ :: ‹*'f set* ⇒ *'f set*› **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*›
+ **assumes**

entails-nontrivial: ‹¬ {} ⊨ {}› **and**

reducedness: ‹*Inf-between UNIV* ($FRed_F$ *N*) ⊆ $FRed_I$ *N*› **and**

complete: ‹*bot* ∉ $FRed_F$ *N*› **and**

all-red-to-bot: ‹$\mathcal{C} \neq bot \Longrightarrow \mathcal{C} \in FRed_F\ \{bot\}$›
begin

notation *sound-cons.entails-neg* (**infix** ‹$\models s_\sim$› *50*)

8.1 The inference rules

Every inference rule X is defined using two functions: X_pre and X_inf. X_inf is the inference rule itself, while X_pre are side-conditions for the rule to be applicable.

abbreviation *base-pre* :: ‹('*f*, '*v*) *AF list* ⇒ '*f* ⇒ *bool*› **where**
‹*base-pre* $\mathcal{N}$ *D* ≡ *Infer* (*map F-of* $\mathcal{N}$) *D* ∈ *FInf*›

abbreviation *base-inf* :: ‹('*f*, '*v*) *AF list* ⇒ '*f* ⇒ ('*f*, '*v*) *AF inference*› **where**
‹*base-inf* $\mathcal{N}$ *D* ≡ *Infer* $\mathcal{N}$ (*Pair D* (*ffUnion* (*fset-of-list* (*map A-of* $\mathcal{N}$))))›

abbreviation *unsat-pre* :: ‹('*f*, '*v*) *AF list* ⇒ *bool*› **where**
‹*unsat-pre* $\mathcal{N}$ ≡ (∀ *x* ∈ *set* $\mathcal{N}$. *F-of x* = *bot*) ∧ *propositionally-unsatisfiable* (*set* $\mathcal{N}$)›

abbreviation *unsat-inf* :: ‹('*f*, '*v*) *AF list* ⇒ ('*f*, '*v*) *AF inference*› **where**
‹*unsat-inf* $\mathcal{N}$ ≡ *Infer* $\mathcal{N}$ (*to-AF bot*)›

We consider first only the inference rules Base and Unsat. The optional inference and simplification rules are handled separately in the locales *splitting-calculus-extensions* and *splitting-calculus-with-simps* respectively.

inductive-set *SInf* :: ‹('*f*, '*v*) *AF inference set*› **where**
base: ‹*base-pre* $\mathcal{N}$ *D* ⟹ *base-inf* $\mathcal{N}$ *D* ∈ *SInf*›
| *unsat*: ‹*unsat-pre* $\mathcal{N}$ ⟹ *unsat-inf* $\mathcal{N}$ ∈ *SInf*›

The predicates in *Splitting-rules* form a valid inference system.

interpretation *SInf-inf-system*: *inference-system SInf* ⟨*proof*⟩

lemma *not-empty-entails-bot*: ‹¬{} ⊨ {*bot*}›
⟨*proof*⟩

The proof for Lemma 13 is split into two parts, for each inclusion in the set equality.

lemma *SInf-commutes-Inf1*:
‹*bot* ∉ $\mathcal{N}$ $proj_J$ *J* ⟹ (*inference-system.Inf-from SInf* $\mathcal{N}$) *ι*$proj_J$ *J* ⊆ *Inf-from* ($\mathcal{N}$ $proj_J$ *J*)›
⟨*proof*⟩

lemma *SInf-commutes-Inf2*:
‹*bot* ∉ $\mathcal{N}$ $proj_J$ *J* ⟹ *Inf-from* ($\mathcal{N}$ $proj_J$ *J*) ⊆ (*inference-system.Inf-from SInf* $\mathcal{N}$) *ι*$proj_J$ *J*›

⟨*proof*⟩

We use *bot* $\notin$ *?𝒩 proj$_J$?J* $\Longrightarrow$ *SInf-inf-system.Inf-from ?𝒩 ιproj$_J$?J* $\subseteq$ *Inf-from* (*?𝒩 proj$_J$?J*) and *bot* $\notin$ *?𝒩 proj$_J$?J* $\Longrightarrow$ *Inf-from* (*?𝒩 proj$_J$?J*) $\subseteq$ *SInf-inf-system.Inf-from ?𝒩 ιproj$_J$?J* to put the Lemma 13 together into a single proof.

lemma *SInf-commutes-Inf*:
‹*bot* $\notin$ *𝒩 proj$_J$ J* $\Longrightarrow$ (*inference-system.Inf-from SInf 𝒩*) *ιproj$_J$ J* = *Inf-from* (*𝒩 proj$_J$ J*)›
⟨*proof*⟩

theorem *SInf-sound-wrt-entails-sound*: ‹*ι$_S$* $\in$ *SInf* $\Longrightarrow$ *set* (*prems-of ι$_S$*) $\models s_{AF}$ {*concl-of ι$_S$*}›
⟨*proof*⟩

The lifted calculus provides a consequence relation and a sound inference system.

interpretation *AF-sound-cons-rel*: *consequence-relation* ‹*to-AF bot*› ‹($\models s_{AF}$)›
⟨*proof*⟩

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf* ‹*to-AF bot*› ‹($\models s_{AF}$)›
⟨*proof*⟩

8.2 The redundancy criterion

definition $SRed_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **where**
‹$SRed_F$ *𝒩* = { *AF.Pair C A* | *C A*. ∀ *𝒥*. *total-strip 𝒥* ⊇ *fset A* ⟶ *C* ∈ $FRed_F$ (*𝒩 proj$_J$ 𝒥*) }
∪ { *AF.Pair C A* | *C A*. ∃ *𝒞* ∈ *𝒩*. *F-of 𝒞* = *C* ∧ *A-of 𝒞* |⊂| *A* }›

definition $SRed_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **where**
‹$SRed_I$ *𝒩* = { *base-inf ℳ 𝒞* | *ℳ 𝒞*. *base-pre ℳ 𝒞* ∧
(∀ *𝒥*. { *base-inf ℳ 𝒞* } *ιproj$_J$ 𝒥* ⊆ $FRed_I$ (*𝒩 proj$_J$ 𝒥*)) }
∪ { *unsat-inf ℳ* | *ℳ*. *unsat-pre ℳ* ∧ *to-AF bot* ∈ *𝒩* }›

lemma *sredI-𝒩-proj-J-subset-redI-proj-J*: ‹*to-AF bot* $\notin$ *𝒩* $\Longrightarrow$ ($SRed_I$ *𝒩*) *ιproj$_J$ J* ⊆ $FRed_I$ (*𝒩 proj$_J$ J*)›
⟨*proof*⟩

lemma *bot-not-in-sredF-𝒩*: ‹*to-AF bot* $\notin$ $SRed_F$ *𝒩*›
⟨*proof*⟩

We need to set things up for the proof of lemma 18. We first restrict $SRed_I$ to Base inferences (under the name $ARed_I$) and show that it is a

redundancy criterion. And then we consider the case of UNSAT inferences separately.

definition $ARed_F :: \langle('f, 'v)\ AF\ set \Rightarrow ('f, 'v)\ AF\ set\rangle$ **where**
$\langle ARed_F\ \mathcal{N} \equiv SRed_F\ \mathcal{N}\rangle$

definition $ARed_I :: \langle('f, 'v)\ AF\ set \Rightarrow ('f, 'v)\ AF\ inference\ set\rangle$ **where**
$\langle ARed_I\ \mathcal{N} \equiv \{\ base\text{-}inf\ \mathcal{M}\ \mathcal{C} \mid \mathcal{M}\ \mathcal{C}.\ base\text{-}pre\ \mathcal{M}\ \mathcal{C} \wedge$
$(\forall\ \mathcal{J}.\ \{\ base\text{-}inf\ \mathcal{M}\ \mathcal{C}\ \}\ \iota proj_J\ \mathcal{J} \subseteq FRed_I\ (\mathcal{N}\ proj_J\ \mathcal{J}))\ \}\rangle$

definition $AInf :: \langle('f, 'v)\ AF\ inference\ set\rangle$ **where**
$\langle AInf \equiv \{\ base\text{-}inf\ \mathcal{N}\ D \mid \mathcal{N}\ D.\ base\text{-}pre\ \mathcal{N}\ D\ \}\rangle$

definition $\mathcal{G}_F :: \langle'v\ total\text{-}interpretation \Rightarrow ('f, 'v)\ AF \Rightarrow 'f\ set\rangle$ **where**
$\langle \mathcal{G}_F\ \mathcal{J}\ \mathcal{C} \equiv \{\mathcal{C}\}\ proj_J\ \mathcal{J}\rangle$

definition $\mathcal{G}_I :: \langle'v\ total\text{-}interpretation \Rightarrow ('f, 'v)\ AF\ inference \Rightarrow 'f\ inference\ set\rangle$
where
$\langle \mathcal{G}_I\ \mathcal{J}\ \iota \equiv \{\iota\}\ \iota proj_J\ \mathcal{J}\rangle$

We define a wellfounded ordering on A-formulas to strengthen $ARed_I$. Basically, $A \leftarrow \mathcal{C} \sqsupset A \leftarrow \mathcal{C}'$ if $\mathcal{C} \subset \mathcal{C}'$.

definition *tiebreaker-order* $:: \langle('f, 'v :: countable)\ AF\ rel\rangle$ **where**
$\langle tiebreaker\text{-}order \equiv \{\ (\mathcal{C}, \mathcal{C}').\ F\text{-}of\ \mathcal{C} = F\text{-}of\ \mathcal{C}' \wedge A\text{-}of\ \mathcal{C} \mid\subset\mid A\text{-}of\ \mathcal{C}'\ \}\rangle$

abbreviation *sqsupset-is-tiebreaker-order* (**infix** $\langle\sqsupset\rangle$ *50*) **where**
$\langle \mathcal{C} \sqsupset \mathcal{C}' \equiv (\mathcal{C}, \mathcal{C}') \in tiebreaker\text{-}order\rangle$

lemma *tiebreaker-order-is-strict-partial-order*: $\langle po\text{-}on\ (\sqsupset)\ UNIV\rangle$
⟨*proof*⟩

lemma *wfp-on-fsubset*: $\langle wfp\text{-}on\ (|\subset|)\ UNIV\rangle$
⟨*proof*⟩

lemma *wfp-on-tiebreaker-order*: $\langle wfp\text{-}on\ (\sqsupset)\ (UNIV :: ('f, 'v)\ AF\ set)\rangle$
⟨*proof*⟩

We can lift inferences from $FRed_I$ to $ARed_I$.

interpretation *lift-from-FRed-to-ARed*: *light-tiebreaker-lifting* $\langle\{to\text{-}AF\ bot\}\rangle$ *AInf* $\langle\{bot\}\rangle$ $\langle(\models\cap)\rangle$
FInf FRed$_I$ *FRed*$_F$ $\langle\mathcal{G}_F\ \mathcal{J}\rangle$ $\langle Some \circ \mathcal{G}_I\ \mathcal{J}\rangle$ $\langle\lambda\text{-}.\ (\sqsupset)\rangle$
⟨*proof*⟩

lemma $ARed_I$*-is-FRed*$_I$: $\langle ARed_I\ \mathcal{N} = (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}$ $J\ \mathcal{N})\rangle$
⟨*proof*⟩

lemma $ARed_F$*-is-FRed*$_F$: $\langle ARed_F\ \mathcal{N} = (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}$ $J\ \mathcal{N})\rangle$

⟨*proof*⟩

lemma *entails-is-entails-G*: ‹$\mathcal{M} \models_{AF} \{\mathcal{C}\} \longleftrightarrow (\forall\ \mathcal{J}.$ *lift-from-FRed-to-ARed.entails-G* $\mathcal{J}\ \mathcal{M}\ \{\mathcal{C}\})$›
⟨*proof*⟩

lemma *SRed$_I$-in-SInf*: ‹*SRed$_I$* $N \subseteq$ *SInf*›
⟨*proof*⟩

lemma *SRed$_F$-entails-bot*: ‹$N \models_{AF} \{$*to-AF bot*$\} \Longrightarrow N -$ *SRed$_F$* $N \models_{AF} \{$*to-AF bot*$\}$›
⟨*proof*⟩

lemma *SRed$_F$-of-subset-F*: ‹$N \subseteq N' \Longrightarrow$ *SRed$_F$* $N \subseteq$ *SRed$_F$* N'›
⟨*proof*⟩

lemma *SRed$_I$-of-subset-F*: ‹$N \subseteq N' \Longrightarrow$ *SRed$_I$* $N \subseteq$ *SRed$_I$* N'›
⟨*proof*⟩

lemma *SRed$_F$-of-SRed$_F$-subset-F*: ‹$N' \subseteq$ *SRed$_F$* $N \Longrightarrow$ *SRed$_F$* $N \subseteq$ *SRed$_F$* $(N - N')$›
⟨*proof*⟩

lemma *SRed$_I$-of-SRed$_F$-subset-F*: ‹$N' \subseteq$ *SRed$_F$* $N \Longrightarrow$ *SRed$_I$* $N \subseteq$ *SRed$_I$* $(N - N')$›
⟨*proof*⟩

lemma *SRed$_I$-of-SInf-to-N-F*: ‹$\iota_S \in$ *SInf* $\Longrightarrow$ *concl-of* $\iota_S \in N \Longrightarrow \iota_S \in$ *SRed$_I$* N›
⟨*proof*⟩

end

8.3 Standard completeness

context *annotated-calculus*
begin

lemmas *SRed-rules* = *SRed$_F$-entails-bot SRed$_F$-of-subset-F SRed$_I$-of-subset-F SRed$_F$-of-SRed$_F$-subset-F SRed$_I$-of-SRed$_F$-subset-F SRed$_I$-of-SInf-to-N-F SRed$_I$-in-SInf*

sublocale *S-calculus*: *calculus* ‹*to-AF bot*› *SInf AF-entails SRed$_I$ SRed$_F$*
⟨*proof*⟩

lemma *S-saturated-to-F-saturated*: ‹*S-calculus.saturated* $\mathcal{N} \Longrightarrow$ *saturated* ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)›
⟨*proof*⟩

notation *AF-cons-rel.entails-conjunctive* (**infix** ‹$\models\cap_{AF}$› *50*)

theorem *S-calculus-statically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) *FRed*$_I$ *FRed*$_F$›
shows ‹*statically-complete-calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) *SRed*$_I$ *SRed*$_F$›
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

The following proof works as follows.

We assume that (*Inf*, (*Red*$_I$, *Red*$_F$)) is statically complete. From that and theorem *Calculi-And-Annotations.statically-complete-calculus bot FInf* ($\models$) *FRed*$_I$ *FRed*$_F$ $\Longrightarrow$ *Calculi-And-Annotations.statically-complete-calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) *SRed*$_I$ *SRed*$_F$, we obtain that (*SInf*, (*SRed*$_I$, *SRed*$_F$)) is statically complete. This means that for all $\mathcal{N} \subseteq$ *UNIV*, if $\mathcal{N}$ is saturated w.r.t. (*SInf*, *SRed*$_I$) and $\mathcal{N} \models\cup_{AF} \{\bot\}$ then $\bot \in \mathcal{N}$. Since $\models\cup_{AF} \equiv \models\cap_{AF}$ when the right hand side is a singleton set, we have that for all $\mathcal{N} \subseteq$ *UNIV*, if $\mathcal{N}$ is saturated w.r.t. (*SInf*, *SRed*$_I$) and $\mathcal{N} \models\cap_{AF} \{\bot\}$ then $\bot \in \mathcal{N}$.

Because $\models\cap_{AF}$ is a consequence relation for the Saturation Framework, we can derive that (*SInf*, (*SRed*$_I$, *SRed*$_F$)) is dynamically complete (using the conjunctive entailment). We then proceed as above but in the opposite way to show that (*SInf*, (*SRed*$_I$, *SRed*$_F$)) is dynamically complete using the disjunctive entailment $\models\cup_{AF}$.

corollary *S-calculus-dynamically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) *FRed*$_I$ *FRed*$_F$›
shows ‹*dynamically-complete-calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) *SRed*$_I$ *SRed*$_F$›
⟨*proof*⟩

8.4 Strong completeness

theorem *S-calculus-strong-statically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) *FRed*$_I$ *FRed*$_F$› **and**
N-locally-saturated: ‹*locally-saturated* $\mathcal{N}$› **and**
N-entails-bot: ‹$\mathcal{N} \models_{AF}$ {*to-AF bot*}›
shows ‹*to-AF bot* $\in \mathcal{N}$›
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

lemma *locally-fair-derivation-is-saturated-at-liminf*:

‹*is-derivation S-calculus.derive* $\mathcal{N}i \Longrightarrow$ *locally-fair* $\mathcal{N}i \Longrightarrow$ *locally-saturated* (*lim-inf* $\mathcal{N}i$)›
⟨*proof*⟩⟨*proof*⟩

theorem *S-calculus-strong-dynamically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) $FRed_I$ $FRed_F$› **and**
Ni-is-derivation: ‹*is-derivation S-calculus.derive* $\mathcal{N}i$› **and**
Ni-is-locally-fair: ‹*locally-fair* $\mathcal{N}i$› **and**
Ni0-entails-bot: ‹*llhd* $\mathcal{N}i \models_{AF} \{$*to-AF bot*$\}$›
shows ‹$\exists$ *i*. *to-AF bot* $\in$ *llnth* $\mathcal{N}i$ *i*›
⟨*proof*⟩

end

9 Extensions: Inferences and simplifications

9.1 Simplifications

datatype *'f simplification* =
Simplify (*S-from*: ‹*'f set*›) (*S-to*: ‹*'f set*›)

Simplification rules are said to be sound if every conclusion is entailed by all premises. We could have also used our conjunctive entailment *consequence-relation.entails-conjunctive*, but it is defined that way so there is nothing to worry about.

locale *AF-calculus* = *sc*: *sound-calculus bot Inf entails entails-sound* Red_I Red_F
for
bot :: (*'f*, *'v* :: *countable*) *AF* **and**
Inf :: ‹(*'f*, *'v*) *AF inference set*› **and**
entails :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool* **and**
entails-sound :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool* **and**
Red_I :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
Red_F :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*

locale *AF-calculus-extended* =
AF-calculus bot Inf entails entails-sound Red_I Red_F
for *bot* :: ‹(*'f*, *'v* :: *countable*) *AF*› **and**
Inf :: ‹(*'f*, *'v*) *AF inference set*› **and**
entails :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*› **and**
entails-sound :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s$› *50*)
and
Red_I :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set*› **and**
Red_F :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*›
+ **fixes**
Simps :: ‹(*'f*, *'v*) *AF simplification set*› **and**
OptInfs :: ‹(*'f*, *'v*) *AF inference set*›

assumes
simps-simp: ‹$\delta \in Simps \Longrightarrow (S\text{-}from\ \delta - S\text{-}to\ \delta) \subseteq Red_F\ (S\text{-}to\ \delta)$› **and**
simps-sound: ‹$\delta \in Simps \Longrightarrow \forall \mathcal{C} \in S\text{-}to\ \delta.\ S\text{-}from\ \delta \models s\ \{\mathcal{C}\}$› **and**

infs-sound: ‹$\iota \in OptInfs \Longrightarrow set\ (prems\text{-}of\ \iota) \models s\ \{concl\text{-}of\ \iota\}$›
begin

lemma *simp-in-derivations*: ‹$\delta \in Simps \Longrightarrow$
sc.derive $(\mathcal{M} \cup S\text{-}from\ \delta)\ (\mathcal{M} \cup S\text{-}to\ \delta)$›
⟨*proof*⟩

lemma *opt-infs-in-derivations*: ‹$\iota \in OptInfs \Longrightarrow$
sc.derive $(\mathcal{M} \cup set\ (prems\text{-}of\ \iota))\ (\mathcal{M} \cup set\ (prems\text{-}of\ \iota) \cup \{concl\text{-}of\ \iota\})$›
⟨*proof*⟩

end

Empty sets of simplifications and optional inferences are accepted in term*locale AF-calculus-extended*

context *AF-calculus*
begin

sublocale *empty-simps*:
AF-calculus-extended bot Inf entails entails-sound Red_I Red_F
$\{\}$ :: $('f, 'v)$ *AF simplification set* $\{\}$ :: $('f, 'v)$ *AF inference set*
⟨*proof*⟩

end

Here we extend our basic calculus with simplification rules, one at a time:

- Split performs a n-ary case analysis on the head of the premise;
- Collect performs garbage collection on clauses which contain propositionally unsatisfiable heads;
- Trim removes assertions which are entailed by others.

9.1.1 The Split Rule

locale *AF-calculus-with-split* =
base-calculus: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$
$SRed_F$ *Simps OptInfs*
for *bot* :: ‹$('f, 'v :: countable)$ *AF*› **and**
SInf :: ‹$('f, 'v)$ *AF inference set*› **and**
entails :: ‹$('f, 'v)$ *AF set* ⇒ $('f, 'v)$ *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹$('f, 'v)$ *AF set* ⇒ $('f, 'v)$ *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF} s$› *50*)
and

$SRed_I$ :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF inference set› **and**
$SRed_F$ :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF set› **and**
Simps :: ‹('f, 'v) AF simplification set› **and**
OptInfs :: ‹('f, 'v) AF inference set›
+ **fixes**
splittable :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ bool›
assumes

split-sound1: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹
{$\mathcal{C}$} $\models_{AF}s$ {*AF.Pair* (*F-of bot*) (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)}› **and**
split-sound2: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ ∀ $\mathcal{C}'$ ∈ *fset* $\mathcal{C}s$. {$\mathcal{C}$} $\models_{AF}s$ {$\mathcal{C}'$}› **and**
split-simp: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ $SRed_F$
({*AF.Pair* (*F-of bot*) (*ffUnion* ((|‘|) *neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)} ∪ *fset* $\mathcal{C}s$)›
begin

Rule definitions follow a similar naming convention to our two inference rules Base and Unsat defined in *annotated-calculus*: *X_simp* is the definition of the simplification rule, while *X_pre* is some precondition which must hold for the rule to be applicable.

abbreviation *split-pre* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ bool› **where**
‹*split-pre* $\mathcal{C}$ $\mathcal{C}s$ ≡ *splittable* $\mathcal{C}$ $\mathcal{C}s$›

abbreviation *split-res* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ ('f, 'v) AF set› **where**
‹*split-res* $\mathcal{C}$ $\mathcal{C}s$ ≡
(*insert* (*AF.Pair* (*F-of bot*) (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)) (*fset* $\mathcal{C}s$))›

abbreviation *split-simp* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ ('f, 'v) AF simplification› **where**
‹*split-simp* $\mathcal{C}$ $\mathcal{C}s$ ≡ *Simplify* {$\mathcal{C}$} (*split-res* $\mathcal{C}$ $\mathcal{C}s$)›

inductive-set *Simps-with-Split* :: ‹('f, 'v) AF simplification set› **where**
split: ‹*split-pre* $\mathcal{C}$ $\mathcal{C}s$ ⟹ *split-simp* $\mathcal{C}$ $\mathcal{C}s$ ∈ *Simps-with-Split*›
| *other*: ‹*simp* ∈ *Simps* ⟹ *simp* ∈ *Simps-with-Split*›

theorem *Inf-with-split-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-Split* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models_{AF}s$ {$\mathcal{C}$}›
⟨*proof*⟩

lemma *split-redundant*: ‹*split-pre* $\mathcal{C}$ $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ $SRed_F$ (*split-res* $\mathcal{C}$ $\mathcal{C}s$)›
⟨*proof*⟩

lemma *simps-with-split-are-simps*: ‹$\iota \in$ *Simps-with-Split* $\Longrightarrow$ (*S-from* ι − *S-to* ι) $\subseteq$ $SRed_F$ (*S-to* ι)›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$ *Simps-with-Split OptInfs*
⟨*proof*⟩

end

locale *splitting-calculus* =
core: *annotated-calculus bot Inf entails entails-sound* Red_I Red_F *fml asn*
for *bot* :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: ‹*'f set* ⇒ *'f set* ⇒ *bool*› (**infix** ‹$\models$› *50*) **and**
entails-sound :: ‹*'f set* ⇒ *'f set* ⇒ *bool*› (**infix** ‹$\models s$› *50*) **and**
Red_I :: ‹*'f set* ⇒ *'f inference set*› **and**
Red_F :: ‹*'f set* ⇒ *'f set*› **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ (*'v* :: *countable*) *sign set*›
begin

interpretation *AF-sound-cons-rel*: *consequence-relation* ‹*to-AF bot*› ‹($\models s_{AF}$)›
⟨*proof*⟩

interpretation *SInf-sound-inf-system*: *sound-inference-system core.SInf* ‹*to-AF bot*› ‹($\models s_{AF}$)›
⟨*proof*⟩

Rule definitions follow a similar naming convention to our two inference rules Base and Unsat defined in *annotated-calculus*: *X_simp* is the definition of the simplification rule, while *X_pre* is some precondition which must hold for the rule to be applicable.

definition *split-form* :: ‹*'f* ⇒ *'f fset* ⇒ *bool*› **where**
‹*split-form C Cs* ⟷ *C* ≠ *bot* ∧ *fcard Cs* ≥ *2*
∧ {*C*} $\models s$ *fset Cs* ∧ (∀ *C'*. *C'* |∈| *Cs* ⟶ *C* ∈ Red_F {*C'*})›

definition *mk-split* :: ‹*'f* ⇒ *'f fset* ⇒ (*'f*, *'v*) *AF fset*› **where**
‹*split-form C Cs* ⟹ *mk-split C Cs* ≡ (λ *C'*. *AF.Pair C'* {| *SOME a*. *a* ∈ *asn* (*Pos C'*) |}) |‘| *Cs*›

definition *splittable* :: ‹(*'f*, *'v*) *AF* ⇒ (*'f*, *'v*) *AF fset* ⇒ *bool*› **where**
‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ≡ *split-form* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) ∧ *mk-split* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) = $\mathcal{C}s$›

lemma *split-creates-singleton-assertion-sets*:
‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ *A* |∈| $\mathcal{C}s$ ⟹ (∃ *a*. *A-of A* = {| *a* |})›
⟨*proof*⟩

lemma *split-all-assertion-sets-asn*:
‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow A$ |∈| $\mathcal{C}s \Longrightarrow (\exists\ a.\ A\text{-}of\ A = \{|a|\} \wedge a \in asn\ (Pos\ (F\text{-}of\ A)))$›
⟨*proof*⟩

lemma *split-all-pairs-in-As-in-Cs*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow (\forall\ P.\ P$ |∈| $\mathcal{C}s \longrightarrow F\text{-}of\ P$ |∈| $(F\text{-}of$ |‘| $\mathcal{C}s))$›
⟨*proof*⟩

lemma *split-all-pairs-in-Cs-in-As*:
‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow (\forall\ C.\ C$ |∈| $(F\text{-}of$ |‘| $\mathcal{C}s) \longrightarrow (\exists\ a.\ AF.Pair\ C\ \{|a|\}$ |∈| $\mathcal{C}s))$›
⟨*proof*⟩

lemma *split-not-empty*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \mathcal{C}s \neq \{||\}$›
⟨*proof*⟩

notation *core.sound-cons.entails-neg* (**infix** ‹$\models s_{\sim}$› *50*)

lemma *split-sound1*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow$
$\{\mathcal{C}\} \models s_{AF}$ $\{AF.Pair\ bot\ (ffUnion\ (fimage\ neg$ |‘| $A\text{-}of$ |‘| $\mathcal{C}s)$ |∪| $A\text{-}of\ \mathcal{C})\}$›
⟨*proof*⟩

lemma *split-sound2*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \forall\ \mathcal{C}' \in fset\ \mathcal{C}s.\ \{\mathcal{C}\} \models s_{AF}\ \{\mathcal{C}'\}$›
⟨*proof*⟩

lemma *split-simp*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \mathcal{C} \in$
$core.SRed_F$ ({ $AF.Pair\ bot\ (ffUnion\ ((|‘|)\ neg$ |‘| $A\text{-}of$ |‘| $\mathcal{C}s)$ |∪| $A\text{-}of\ \mathcal{C})$ } ∪ $fset\ \mathcal{C}s)$›
⟨*proof*⟩

sublocale *empty-simps*: *AF-calculus-extended to-AF bot core.SInf* $(\models_{AF})$
$(\models s_{AF})$ $core.SRed_I$ $core.SRed_F$ {} {}
⟨*proof*⟩

lemma *extend-simps-with-split*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* $(\models_{AF})$ $(\models s_{AF})$ $core.SRed_I$
$core.SRed_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-split* (*to-AF bot*) *core.SInf* $(\models_{AF})$ $(\models s_{AF})$ $core.SRed_I$ $core.SRed_F$
Simps OptInfs
splittable›
⟨*proof*⟩

interpretation *splitting-calc-with-split*:
AF-calculus-with-split to-AF bot core.SInf $(\models_{AF})$ $(\models s_{AF})$ $core.SRed_I$ $core.SRed_F$
{} {} *splittable*

⟨*proof*⟩

end

9.1.2 The Collect Rule

locale *AF-calculus-with-collect* = *base-calculus*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
$SRed_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
$SRed_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
+ **assumes**
collect-redundant: ‹*F-of* $\mathcal{C}$ ≠ *F-of bot* ∧ $\mathcal{M} \models_{AF}$ {*AF.Pair* (*F-of bot*) (*A-of* $\mathcal{C}$)} ∧
(∀ $\mathcal{C} \in \mathcal{M}$. *F-of* $\mathcal{C}$ = *F-of bot*) ⟹ $\mathcal{C} \in SRed_F\ \mathcal{M}$›
begin

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹($\models s_{AF}$)›
⟨*proof*⟩

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹($\models s_{AF}$)›
⟨*proof*⟩

abbreviation *collect-pre* :: ‹('*f*, '*v*) *AF* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **where**
‹*collect-pre* $\mathcal{C}$ $\mathcal{M}$ ≡
F-of $\mathcal{C}$ ≠ *F-of bot* ∧ $\mathcal{M} \models_{AF}$ {*AF.Pair* (*F-of bot*) (*A-of* $\mathcal{C}$)} ∧ (∀ $\mathcal{C} \in \mathcal{M}$. *F-of* $\mathcal{C}$ = *F-of bot*)›

abbreviation *collect-simp* :: ‹('*f*, '*v*) *AF* ⇒ ('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF simplification*› **where**
‹*collect-simp* $\mathcal{C}$ $\mathcal{M}$ ≡ *Simplify* (*insert* $\mathcal{C}$ $\mathcal{M}$) $\mathcal{M}$›

inductive-set *Simps-with-Collect* :: ‹('*f*, '*v*) *AF simplification set*› **where**
collect: ‹*collect-pre* $\mathcal{C}$ $\mathcal{M}$ ⟹ *collect-simp* $\mathcal{C}$ $\mathcal{M}$ ∈ *Simps-with-Collect*›
| *other*: ‹ι ∈ *Simps* ⟹ ι ∈ *Simps-with-Collect*›

theorem *SInf-with-collect-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-Collect* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
⟨*proof*⟩

theorem *simps-with-collect-are-simps*:
‹$\iota \in$ *Simps-with-Collect* $\Longrightarrow$ (*S-from* ι − *S-to* ι) $\subseteq$ $SRed_F$ (*S-to* ι)›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$
Simps-with-Collect OptInfs
⟨*proof*⟩

end

context *splitting-calculus*
begin

lemma *collect-redundant*: ‹*F-of* $\mathcal{C} \neq$ *bot* $\wedge$ $\mathcal{M} \models_{AF}$ {*AF.Pair bot* (*A-of* $\mathcal{C}$)} $\wedge$
(∀ $\mathcal{C} \in \mathcal{M}$. *F-of* $\mathcal{C}$ = *bot*) $\Longrightarrow$ $\mathcal{C} \in$ *core.*$SRed_F$ $\mathcal{M}$›
⟨*proof*⟩

lemma *extend-simps-with-collect*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.*$SRed_I$
core.$SRed_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-collect* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.*$SRed_I$ *core.*$SRed_F$ *Simps OptInfs*›
⟨*proof*⟩

interpretation *splitting-calc-with-collect*:
AF-calculus-with-collect to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.*$SRed_I$ *core.*$SRed_F$ {} {}
⟨*proof*⟩

end

9.1.3 The Trim Rule

locale *AF-calculus-with-trim* = *base-calculus*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*) **and**
$SRed_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
$SRed_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**

Simps :: ‹('f, 'v) *AF simplification set*› **and**
OptInfs :: ‹('f, 'v) *AF inference set*›
+ **assumes**
trim-sound: ‹*A-of* $\mathcal{C}$ = *A* |∪| *B* ∧ *F-of* $\mathcal{C}$ ≠ *F-of bot* ∧
$\mathcal{M}$ ∪ {*AF.Pair* (*F-of bot*) *A*} ⊨s_{AF} {*AF.Pair* (*F-of bot*) *B*} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. *F-of* $\mathcal{C}$ = (*F-of bot*)) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||}
⟹ *insert* $\mathcal{C}$ $\mathcal{M}$ ⊨s_{AF} {*AF.Pair* (*F-of* $\mathcal{C}$) *B*}› **and**
trim-redundant: ‹*A-of* $\mathcal{C}$ = *A* |∪| *B* ∧ *F-of* $\mathcal{C}$ ≠ *F-of bot* ∧
$\mathcal{M}$ ∪ {*AF.Pair* (*F-of bot*) *A*} ⊨s_{AF} {*AF.Pair* (*F-of bot*) *B*} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. *F-of* $\mathcal{C}$ = (*F-of bot*)) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||}
⟹ $\mathcal{C}$ ∈ *SRed*$_F$ ($\mathcal{M}$ ∪ { *AF.Pair* (*F-of* $\mathcal{C}$) *B* })›

begin

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹(⊨s_{AF})›
⟨*proof*⟩

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹(⊨s_{AF})›
⟨*proof*⟩

abbreviation *trim-pre* :: ‹('f, 'v) *AF* ⇒ 'v *sign fset* ⇒ 'v *sign fset* ⇒ ('f, 'v) *AF set* ⇒ *bool*› **where**
‹*trim-pre* $\mathcal{C}$ *A B* $\mathcal{M}$ ≡ *A-of* $\mathcal{C}$ = *A* |∪| *B* ∧ *F-of* $\mathcal{C}$ ≠ *F-of bot* ∧
$\mathcal{M}$ ∪ { *AF.Pair* (*F-of bot*) *A* } ⊨s_{AF} { *AF.Pair* (*F-of bot*) *B* } ∧
(∀ $\mathcal{C}$ ∈ $\mathcal{M}$. *F-of* $\mathcal{C}$ = (*F-of bot*)) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||}›

abbreviation *trim-simp* :: ‹('f, 'v) *AF* ⇒ 'v *sign fset* ⇒ 'v *sign fset* ⇒ ('f, 'v) *AF set* ⇒
('f, 'v) *AF simplification*› **where**
‹*trim-simp* $\mathcal{C}$ *A B* $\mathcal{M}$ ≡ *Simplify* (*insert* $\mathcal{C}$ $\mathcal{M}$) (*insert* (*AF.Pair* (*F-of* $\mathcal{C}$) *B*) $\mathcal{M}$)›

inductive-set *Simps-with-Trim* :: ‹('f, 'v) *AF simplification set*› **where**
trim: ‹*trim-pre* $\mathcal{C}$ *A B* $\mathcal{M}$ ⟹ *trim-simp* $\mathcal{C}$ *A B* $\mathcal{M}$ ∈ *Simps-with-Trim*›
| *other*: ‹ι ∈ *Simps* ⟹ ι ∈ *Simps-with-Trim*›

theorem *SInf-with-trim-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-Trim* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι ⊨s_{AF} {$\mathcal{C}$}›
⟨*proof*⟩

theorem *simps-with-trim-are-simps*:
‹ι ∈ *Simps-with-Trim* ⟹ (*S-from* ι − *S-to* ι) ⊆ *SRed*$_F$ (*S-to* ι)›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$
Simps-with-Trim OptInfs
⟨*proof*⟩

end

context *splitting-calculus*
begin

lemma *trim-sound*: ‹*A-of* $\mathcal{C}' = A \mid\cup\mid B \wedge$ *F-of* $\mathcal{C}' \neq$ *bot* $\wedge$
$\mathcal{N} \cup \{AF.Pair\ bot\ A\} \models s_{AF} \{AF.Pair\ bot\ B\} \wedge$
$(\forall \mathcal{C} \in \mathcal{N}.$ *F-of* $\mathcal{C} =$ *bot*$) \wedge A \mid\cap\mid B = \{||\} \wedge A \neq \{||\}$
$\Longrightarrow \mathcal{C} = AF.Pair$ (*F-of* $\mathcal{C}'$) $B \Longrightarrow$ *insert* $\mathcal{C}'\ \mathcal{N} \models s_{AF} \{\mathcal{C}\}$›
⟨*proof*⟩

theorem *trim-redundant*: ‹*A-of* $\mathcal{C} = A \mid\cup\mid B \wedge$
F-of $\mathcal{C} \neq$ *bot* $\wedge$
$\mathcal{M} \cup \{AF.Pair\ bot\ A\} \models s_{AF} \{AF.Pair\ bot\ B\} \wedge$
$(\forall \mathcal{C} \in \mathcal{M}.$ *F-of* $\mathcal{C} =$ *bot*$) \wedge A \mid\cap\mid B = \{||\} \wedge A \neq \{||\} \Longrightarrow$
$\mathcal{C} \in$ *core.SRed*$_F$ ($\mathcal{M} \cup \{$ *AF.Pair* (*F-of* $\mathcal{C}$) B $\}$)›
⟨*proof*⟩

lemma *extend-simps-with-trim*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-trim* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
⟨*proof*⟩

interpretation *splitting-calc-with-trim*:
AF-calculus-with-trim to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
⟨*proof*⟩

end

9.2 Extra Inferences

We extend our basic splitting calculus with new optional rules:

- StrongUnsat is a variant of Unsat which uses the sound entailment instead of the "normal" entailment;
- Approx is a very special case for Split where $n = 1$;
- Tauto inserts a new formula which is always true.

9.2.1 The StrongUnsat Rule

locale *AF-calculus-with-strong-unsat* =
  base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps OptInfs*
  for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
    SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
    entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **and**
    entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
    Red-I :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
    Red-F :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
    Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
    OptInfs :: ‹('*f*, '*v*) *AF inference set*›
begin

We follow the same naming conventions for our new inference rules as for the two inference rules defined in *annotated-calculus*. X_inf defines the inference rule, while X_pre is the precondition for the application of the inference rule.

abbreviation *strong-unsat-pre* :: ‹('*f*, '*v*) *AF list* ⇒ *bool*› **where**
  ‹*strong-unsat-pre* $\mathcal{M}$ ≡ (*set* $\mathcal{M}$ $\models s_{AF}$ {*bot*}) ∧ (∀ *x* ∈ *set* $\mathcal{M}$. *F-of x* = (*F-of bot*))›

abbreviation *strong-unsat-inf* :: ‹('*f*, '*v*) *AF list* ⇒ ('*f*, '*v*) *AF inference*› **where**
  ‹*strong-unsat-inf* $\mathcal{M}$ ≡ *Infer* $\mathcal{M}$ *bot*›

Instead of considering an inference system with only the new rule, here StrongUnsat, we instead add it to the inference system provided so that it is possible to extend the system rule by rule in a modular way.

inductive-set *OptInfs-with-strong-unsat* :: ‹('*f*, '*v*) *AF inference set*› **where**
  strong-unsat: ‹*strong-unsat-pre* $\mathcal{M}$ ⟹ *strong-unsat-inf* $\mathcal{M}$ ∈ *OptInfs-with-strong-unsat*›
| *from-OptInf*: ‹ι ∈ *OptInfs* ⟹ ι ∈ *OptInfs-with-strong-unsat*›

theorem *OptInf-with-strong-unsat-sound-wrt-entails-sound*:
  ‹ι ∈ *OptInfs-with-strong-unsat* ⟹ *set* (*prems-of* ι) $\models s_{AF}$ {*concl-of* ι}›
⟨*proof*⟩

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹($\models s_{AF}$)›
  ⟨*proof*⟩

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹($\models s_{AF}$)›
  ⟨*proof*⟩

definition *Red-I-ext* **where**
  ‹*Red-I-ext* $\mathcal{N}$ = *Red-I* $\mathcal{N}$ ∪ { *strong-unsat-inf* $\mathcal{M}$ | $\mathcal{M}$. *strong-unsat-pre* $\mathcal{M}$ ∧ *bot* ∈ $\mathcal{N}$ }›

interpretation *AF-calc-with-strong-unsat*:
AF-calculus bot SInf entails entails-sound Red-I Red-F
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F*
Simps OptInfs-with-strong-unsat
⟨*proof*⟩

end

context *splitting-calculus*
begin

lemma *extend-infs-with-strong-unsat*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) $core.SRed_I$ $core.SRed_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-strong-unsat* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) $core.SRed_I$ $core.SRed_F$ *Simps OptInfs*›
⟨*proof*⟩

interpretation *splitting-calc-with-strong-unsat*: *AF-calculus-with-strong-unsat to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) $core.SRed_I$ $core.SRed_F$ {} {}
⟨*proof*⟩

end

9.2.2 The Tauto Rule

locale *AF-calculus-with-tauto* =
base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
Red-I :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
Red-F :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
begin

abbreviation *tauto-pre* :: ‹('*f*, '*v*) *AF* ⇒ *bool*› **where**
‹*tauto-pre* $\mathcal{C}$ ≡ {} $\models s_{AF}$ { $\mathcal{C}$ }›

abbreviation *tauto-inf* :: ‹('f, 'v) *AF* ⇒ ('f, 'v) *AF inference*› **where**
‹*tauto-inf* $\mathcal{C}$ ≡ *Infer* [] $\mathcal{C}$›

inductive-set *OptInfs-with-tauto* :: ‹('f, 'v) *AF inference set*› **where**
tauto: ‹*tauto-pre* $\mathcal{C}$ ⟹ *tauto-inf* $\mathcal{C}$ ∈ *OptInfs-with-tauto*›
| *from-OptInfs*: ‹ι ∈ *OptInfs* ⟹ ι ∈ *OptInfs-with-tauto*›

theorem *OptInfs-with-tauto-sound-wrt-entails-sound*: ‹ι ∈ *OptInfs-with-tauto* ⟹

set (*prems-of* ι) $\models s_{AF}$ {*concl-of* ι}›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails* ($\models s_{AF}$) *Red-I*
Red-F Simps OptInfs-with-tauto
⟨*proof*⟩

end

context *splitting-calculus*
begin

lemma *extend-infs-with-tauto*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-tauto* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$
Simps OptInfs›
⟨*proof*⟩

interpretation *splitting-calc-with-tauto*:
AF-calculus-with-tauto to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$
{} {}
⟨*proof*⟩

end

9.2.3 The Approx Rule

locale *AF-calculus-with-approx* =
base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps*
OptInfs
for *bot* :: ‹('f, 'v :: *countable*) *AF*› **and**
SInf :: ‹('f, 'v) *AF inference set*› **and**

entails :: ‹('f, 'v) *AF set* ⇒ ('f, 'v) *AF set* ⇒ *bool*› **and**
entails-sound :: ‹('f, 'v) *AF set* ⇒ ('f, 'v) *AF set* ⇒ *bool*› (**infix** ‹$\models_{s_{AF}}$› *50*)
and
Red-I :: ‹('f, 'v) *AF set* ⇒ ('f, 'v) *AF inference set*› **and**
Red-F :: ‹('f, 'v) *AF set* ⇒ ('f, 'v) *AF set*› **and**
Simps :: ‹('f, 'v) *AF simplification set*› **and**
OptInfs :: ‹('f, 'v) *AF inference set*›
+ **fixes**
approximates :: ‹'v *sign* ⇒ ('f, 'v) *AF* ⇒ *bool*›
assumes
approx-sound: ‹*approximates a* $\mathcal{C}$ ⟹ {$\mathcal{C}$} $\models_{s_{AF}}$ {*AF.Pair* (*F-of bot*) (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
begin

abbreviation *approx-pre* :: ‹'v *sign* ⇒ ('f, 'v) *AF* ⇒ *bool*› **where**
‹*approx-pre a* $\mathcal{C}$ ≡ *approximates a* $\mathcal{C}$›

abbreviation *approx-inf* :: ‹('f, 'v) *AF* ⇒ 'v *sign* ⇒ ('f, 'v) *AF inference*› **where**
‹*approx-inf* $\mathcal{C}$ *a* ≡ *Infer* [$\mathcal{C}$] (*AF.Pair* (*F-of bot*) (*finsert* (*neg a*) (*A-of* $\mathcal{C}$)))›

inductive-set *OptInfs-with-approx* :: ‹('f, 'v) *AF inference set*› **where**
approx: ‹*approx-pre a* $\mathcal{C}$ ⟹ *approx-inf* $\mathcal{C}$ *a* ∈ *OptInfs-with-approx*›
| *from-OptInfs*: ‹ι ∈ *OptInfs* ⟹ ι ∈ *OptInfs-with-approx*›

theorem *OptInfs-with-approx-sound-wrt-entails-sound*:
‹ι ∈ *OptInfs-with-approx* ⟹ *set* (*prems-of* ι) $\models_{s_{AF}}$ {*concl-of* ι}›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails* ($\models_{s_{AF}}$) *Red-I Red-F Simps*
OptInfs-with-approx
⟨*proof*⟩

end

context *splitting-calculus*
begin

definition *approximates* :: ‹'v *sign* ⇒ ('f, 'v) *AF* ⇒ *bool*› **where**
‹*approximates a* $\mathcal{C}$ ≡ *a* ∈ *asn* (*Pos* (*F-of* $\mathcal{C}$))›

lemma *approx-sound*: ‹*approximates a* $\mathcal{C}$ ⟹ {$\mathcal{C}$} $\models_{s_{AF}}$ {*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
⟨*proof*⟩

lemma *extend-infs-with-approx*:
assumes

‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs›
shows
‹*AF-calculus-with-approx* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs approximates›
⟨*proof*⟩

interpretation *splitting-calc-with-approx*:
AF-calculus-with-approx to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
approximates
⟨*proof*⟩

end

9.3 Combining all simplifications and optional inferences

We have augmented the core calculus with each simplification and optional rule separately. We now show how to augment the core calculus with all of them in succession.

context *splitting-calculus*
begin

interpretation *with-A*: *AF-calculus-with-approx to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} {} *approximates*
⟨*proof*⟩

interpretation *with-AT*: *AF-calculus-with-tauto to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} *with-A.OptInfs-with-approx*
⟨*proof*⟩

interpretation *with-ATS*: *AF-calculus-with-strong-unsat to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)
core.SRed$_I$ *core.SRed*$_F$ {} *with-AT.OptInfs-with-tauto*
⟨*proof*⟩

interpretation *with-ATS-T*: *AF-calculus-with-trim to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} *with-ATS.OptInfs-with-strong-unsat*
⟨*proof*⟩

interpretation *with-ATS-TC*: *AF-calculus-with-collect to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)
core.SRed$_I$ *core.SRed*$_F$ *with-ATS-T.Simps-with-Trim with-ATS.OptInfs-with-strong-unsat*
⟨*proof*⟩

interpretation *with-all*: *AF-calculus-with-split to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)

core.SRed$_I$ *core.SRed*$_F$ *with-ATS-TC.Simps-with-Collect with-ATS.OptInfs-with-strong-unsat splittable*
⟨*proof*⟩

interpretation *full-splitting-calculus*: *AF-calculus-extended to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *with-all.Simps-with-Split with-ATS.OptInfs-with-strong-unsat*
⟨*proof*⟩

Simplifications and optional inferences can be integrated in derivations. This is made obvious by the following two lemmas.

lemma *simp-in-derivations*: ‹$\delta \in$ *with-all.Simps-with-Split* $\Longrightarrow$ *core.S-calculus.derive* ($\mathcal{M} \cup$ *S-from* δ) ($\mathcal{M} \cup$ *S-to* δ)›
⟨*proof*⟩

lemma *opt-infs-in-derivations*: ‹$\iota \in$ *with-ATS.OptInfs-with-strong-unsat* $\Longrightarrow$ *core.S-calculus.derive* ($\mathcal{M} \cup$ *set* (*prems-of* ι)) ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
⟨*proof*⟩

end

9.4 The BinSplit Simplification Rule

For the sake of the Lightweight Avatar calculus, we define the BinSplit simplification rule, and show that it is indeed a sound simplification rule as in the case of the Split rule.

locale *AF-calculus-with-binsplit* =
base-calculus: *AF-calculus-extended bot SInf entails entails-sound SRed*$_I$ *SRed*$_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
SRed$_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
SRed$_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
+ **fixes**
bin-splittable :: ‹('*f*, '*v*) *AF* ⇒ '*f* ⇒ '*f* ⇒ ('*f*, '*v*) *AF fset* ⇒ *bool*›
assumes
binsplit-prem-entails-cons: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \forall\ \mathcal{C}' \in$ *fset* $\mathcal{C}s$. {$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}'$}› **and**
binsplit-simp: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \mathcal{C} \in$ *SRed*$_F$ (*fset* $\mathcal{C}s$)›
begin

We use the same naming convention as for the other rules, where X_pre is the condition which must hold for the rule to be applicable, X_res is its resultant and X_simp is the simplification rule itself.

abbreviation *bin-split-pre* :: ‹('f, 'v) *AF* ⇒ 'f ⇒ 'f ⇒ ('f, 'v) *AF fset* ⇒ *bool*›
where
‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ≡ *bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$›

abbreviation *bin-split-res* :: ‹('f, 'v) *AF* ⇒ ('f, 'v) *AF fset* ⇒ ('f, 'v) *AF set*›
where
‹*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$ ≡ *fset* $\mathcal{C}s$›

abbreviation *bin-split-simp* :: ‹('f, 'v) *AF* ⇒ ('f, 'v) *AF fset* ⇒ ('f, 'v) *AF simplification*› **where**
‹*bin-split-simp* $\mathcal{C}$ $\mathcal{C}s$ ≡ *Simplify* {$\mathcal{C}$} (*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$)›

inductive-set *Simps-with-BinSplit* :: ‹('f, 'v) *AF simplification set*› **where**
binsplit: ‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ *bin-split-simp* $\mathcal{C}$ $\mathcal{C}s$ ∈ *Simps-with-BinSplit*›

| *other*: ‹*simp* ∈ *Simps* ⟹ *simp* ∈ *Simps-with-BinSplit*›

theorem *SInf-with-binsplit-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-BinSplit* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
⟨*proof*⟩

lemma *binsplit-redundant*: ‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ $SRed_F$ (*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$)›
⟨*proof*⟩

lemma *simps-with-binsplit-are-simps*:
‹ι ∈ *Simps-with-BinSplit* ⟹ (*S-from* ι − *S-to* ι) ⊆ $SRed_F$ (*S-to* ι)›
⟨*proof*⟩

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound*
$SRed_I$ $SRed_F$ *Simps-with-BinSplit OptInfs*
⟨*proof*⟩

end

context *splitting-calculus*
begin

definition *mk-bin-split* :: ‹('f, 'v) *AF* ⇒ 'f ⇒ 'f ⇒ ('f, 'v) *AF fset*› **where**
‹*split-form* (*F-of* $\mathcal{C}$) {|*C1*, *C2*|} ⟹ *mk-bin-split* $\mathcal{C}$ *C1 C2* ≡
let a = (*SOME a*. *a* ∈ *asn* (*sign.Pos C1*))

in {|*AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)), *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))|}›

definition *bin-splittable* :: ‹('*f*, '*v*) *AF* ⇒ '*f* ⇒ '*f* ⇒ ('*f*, '*v*) *AF fset* ⇒ *bool*›
where
‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ≡ *split-form* (*F-of* $\mathcal{C}$) {|*C1*, *C2*|} ∧ *mk-bin-split* $\mathcal{C}$ *C1 C2* = $\mathcal{C}s$›

theorem *binsplit-prem-entails-cons*: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ ∀ $\mathcal{C}'$ ∈ *fset* $\mathcal{C}s$. {$\mathcal{C}$} ⊨s_{AF} {$\mathcal{C}'$}›
⟨*proof*⟩

theorem *binsplit-simp*: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ *core.SRed*$_F$ (*fset* $\mathcal{C}s$)›
⟨*proof*⟩

lemma *extend-simps-with-binsplit*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* (⊨$_{AF}$) (⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-binsplit* (*to-AF bot*) *core.SInf* (⊨$_{AF}$) (⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs bin-splittable›
⟨*proof*⟩

interpretation *splitting-calc-with-binsplit*: *AF-calculus-with-binsplit to-AF bot core.SInf* (⊨$_{AF}$)
(⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ {} {} *bin-splittable*
⟨*proof*⟩

end

end

theory *Lightweight-Avatar*
imports
Main
Modular-Splitting-Calculus
Saturation-Framework-Extensions.FO-Ordered-Resolution-Prover-Revisited
begin

9.5 Ordered Resolution with a Disjunctive Consequence Relation

locale *FO-resolution-prover-disjunctive* = *FO-resolution-prover S subst-atm id-subst comp-subst*
renaming-aparts atm-of-atms mgu less-atm

for
S :: ‹('*a* :: *wellorder*) *clause* ⇒ '*a clause*› **and**
subst-atm :: ‹'*a* ⇒ '*s* ⇒ '*a*› **and**
id-subst :: ‹'*s*› **and**
comp-subst :: ‹'*s* ⇒ '*s* ⇒ '*s*› **and**
renaming-aparts :: ‹'*a clause list* ⇒ '*s list*› **and**
atm-of-atms :: ‹'*a list* ⇒ '*a*› **and**
mgu :: ‹'*a set set* ⇒ '*s option*› **and**
less-atm :: ‹'*a* ⇒ '*a* ⇒ *bool*›
begin

no-notation *entails-clss* (**infix** ‹⊨*e*› *50*)
no-notation *Sema.entailment* (‹(- ⊨/ -)› [*53*, *53*] *53*)
no-notation *Linear-Temporal-Logic-on-Streams.HLD-nxt* (**infixr** · *65*)

notation *entails-clss* (**infix** ‹⊨∩*e*› *50*)

interpretation *gr*: *ground-resolution-with-selection S-M S M*
⟨*proof*⟩

interpretation *G*: *Soundness.sound-inference-system G-Inf M* {{#}} (⊨∩*e*)
⟨*proof*⟩

interpretation *G*: *clausal-counterex-reducing-inference-system G-Inf M gr.INTERP M*
⟨*proof*⟩

interpretation *G*: *calculus-with-standard-redundancy* ‹*G-Inf M*› ‹{{#}}› ‹(⊨∩*e*)›
‹(<) :: '*a clause* ⇒ '*a clause* ⇒ *bool*›
⟨*proof*⟩

interpretation *G*: *clausal-counterex-reducing-calculus-with-standard-redundancy G-Inf M*
gr.INTERP M
⟨*proof*⟩

interpretation *G*: *Calculus.statically-complete-calculus* {{#}} *G-Inf M* (⊨∩*e*) *G.Red-I M G.Red-F*
⟨*proof*⟩

sublocale *F*: *lifting-intersection F-Inf* {{#}} *UNIV G-Inf* λ*N*. (⊨∩*e*) *G.Red-I* λ*N*. *G.Red-F*
{{#}} λ*N*. *G-F G-I-opt* λ*D C C'*. *False*
⟨*proof*⟩

notation *F.entails-G* (**infix** ⊨∩*G e* *50*)

sublocale F: *sound-inference-system F-Inf* $\{\{\#\}\}$ $(\models\!\!\models\cap\mathcal{G}e)$
⟨*proof*⟩

lemma *F-stat-comp-calc*: ‹*Calculus.statically-complete-calculus* $\{\{\#\}\}$ *F-Inf* $(\models\!\!\models\cap\mathcal{G}e)$ *F.Red-I-*$\mathcal{G}$ *F.Red-F-*$\mathcal{G}$*-empty*›
⟨*proof*⟩

sublocale F: *Calculus.statically-complete-calculus* $\{\{\#\}\}$ *F-Inf* $(\models\!\!\models\cap\mathcal{G}e)$ *F.Red-I-*$\mathcal{G}$ *F.Red-F-*$\mathcal{G}$*-empty*
⟨*proof*⟩

sublocale F': *Calculus.statically-complete-calculus* $\{\{\#\}\}$ *F-Inf* $(\models\!\!\models\cap\mathcal{G}e)$ *F.empty-ord.Red-Red-I F.Red-F-*$\mathcal{G}$*-empty*
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

$(\models\!\!\models\cap\mathcal{G}e)$ is a conjunctive entailment, meaning that for $M \models\!\!\models\cap\mathcal{G}e\ N$ to hold, each clause in N must be entailed by M. Unfortunately, this clashes with requirement (D3) ⟦*Disjunctive-Consequence-Relations.consequence-relation ?bot ?entails*; $?M' \subseteq ?M$; $?N' \subseteq ?N$; *?entails ?M' ?N'*⟧ ⟹ *?entails ?M ?N* of a splitting calculus.

Therefore, we define a disjunctive version of this entailment by stating that $M \models\!\!\models\cup\mathcal{G}e\ N$ iff there is some $C \in N$ such that $M \models\!\!\models\cap\mathcal{G}e\ \{C\}$. This definition is not quite enough because it does not capture (D1) *Disjunctive-Consequence-Relations.consequence-relation ?bot ?entails* ⟹ *?entails* {*?bot*} {}. More specifically, if N is empty, then there does not exist a $C \in N$! But we know that $M \models\cup\mathcal{G}e\ \{\}$ if M is unsatisfiable. Hence $M \models\!\!\models\cup\mathcal{G}e\ N$ if M is unsatisfiable, or there exists some $C \in N$ such that $M \models\!\!\models\cap\mathcal{G}e\ \{C\}$. In addition, it is necessary to modify this definition to capture (D5) the compactness property of disjunctive entailment.

definition *entails-*$\mathcal{G}$*-disj* :: ‹*'a clause set* ⇒ *'a clause set* ⇒ *bool*› (**infix** ‹$\models\!\!\models\cup\mathcal{G}e$›
50) **where**
‹$M \models\!\!\models\cup\mathcal{G}e\ N \longleftrightarrow M \models\!\!\models\cap\mathcal{G}e\ \{\{\#\}\} \vee (\exists\ M' \subseteq M.$ *finite* $M' \wedge (\exists\ C \in N.\ M' \models\!\!\models\cap\mathcal{G}e\ \{C\}))$›

⟨*proof*⟩

lemma *entails-*$\mathcal{G}$*-disj-subsets*: ‹$M' \subseteq M \Longrightarrow N' \subseteq N \Longrightarrow M' \models\!\!\models\cup\mathcal{G}e\ N' \Longrightarrow M \models\!\!\models\cup\mathcal{G}e\ N$›
⟨*proof*⟩

lemma *entails-*$\mathcal{G}$*-disj-compactness*:
‹$M \models\!\!\models\cup\mathcal{G}e\ N \Longrightarrow \exists\ M'\ N'.\ M' \subseteq M \wedge N' \subseteq N \wedge$ *finite* $M' \wedge$ *finite* $N' \wedge M' \models\!\!\models\cup\mathcal{G}e\ N'$›
⟨*proof*⟩

lemma *entails-$\mathcal{G}$-disj-cut*: ‹$M \Vdash\cup\mathcal{G}e\ N \cup \{C\} \Longrightarrow M' \cup \{C\} \Vdash\cup\mathcal{G}e\ N' \Longrightarrow M \cup M' \Vdash\cup\mathcal{G}e\ N \cup N'$›
⟨*proof*⟩

lemma *entails-$\mathcal{G}$-disj-cons-rel-ext*: ‹*consequence-relation* $\{\#\}$ $(\Vdash\cup\mathcal{G}e)$›
⟨*proof*⟩

sublocale *entails-$\mathcal{G}$-disj-cons-rel*: *consequence-relation* ‹$\{\#\}$› ‹$(\Vdash\cup\mathcal{G}e)$›
⟨*proof*⟩

notation *entails-$\mathcal{G}$-disj-cons-rel.entails-neg* (**infix** ‹$\Vdash\cup\mathcal{G}e_\sim$› *50*)

lemma *all-redundant-to-bottom*: ‹$\mathcal{C} \neq \{\#\} \Longrightarrow \mathcal{C} \in$ *F.Red-F-$\mathcal{G}$-empty* $\{\{\#\}\}$›
⟨*proof*⟩

lemma *bottom-never-redundant*: ‹$\{\#\} \notin$ *F.Red-F-$\mathcal{G}$-empty N*›
⟨*proof*⟩

lemma ‹*F.Inf-between UNIV* (*F.Red-F-$\mathcal{G}$-empty N*) $\subseteq$ *F.empty-ord.Red-Red-I N*›
⟨*proof*⟩

end

9.6 Lightweight Avatar without BinSplit

Since the set $\mathbb{P}$ of nullary predicates is left unspecified, we cannot define *fml* nor *asn*. Therefore, we keep them abstract and leave it to anybody instantiating this locale to specify them.

locale *LA-calculus* = *FO-resolution-prover-disjunctive S subst-atm id-subst comp-subst renaming-aparts*
atm-of-atms mgu less-atm
for
S :: ‹(*'a* :: *wellorder*) *clause* ⇒ *'a clause*› **and**
subst-atm :: ‹*'a* ⇒ *'s* ⇒ *'a*› **and**
id-subst :: ‹*'s*› **and**
comp-subst :: ‹*'s* ⇒ *'s* ⇒ *'s*› **and**
renaming-aparts :: ‹*'a clause list* ⇒ *'s list*› **and**
atm-of-atms :: ‹*'a list* ⇒ *'a*› **and**
mgu :: ‹*'a set set* ⇒ *'s option*› **and**
less-atm :: ‹*'a* ⇒ *'a* ⇒ *bool*›
+ **fixes**
asn :: ‹*'a clause sign* ⇒ (*'v* :: *countable*) *sign set*› **and**
fml :: ‹*'v* ⇒ *'a clause*›
assumes
asn-not-empty: ‹*asn C* ≠ {}› **and**
fml-entails-C: ‹*a* ∈ *asn C* ⟹ {*map-sign fml a*} $\Vdash\cup\mathcal{G}e_\sim$ {*C*}› **and**
C-entails-fml: ‹*a* ∈ *asn C* ⟹ {*C*} $\Vdash\cup\mathcal{G}e_\sim$ {*map-sign fml a*}›
begin

interpretation *entails-$\mathcal{G}$-disj-sound-inf-system*:
Calculi-And-Annotations.sound-inference-system F-Inf ‹{#}› ‹(⊫∪$\mathcal{G}$ e)›
⟨*proof*⟩

interpretation *LA-is-calculus*: *calculus* ‹{#}› *F-Inf* ‹(⊫∪$\mathcal{G}$ e)› *F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty*
⟨*proof*⟩

interpretation *LA-is-sound-calculus*: *sound-calculus* ‹{#}› *F-Inf* ‹(⊫∪$\mathcal{G}$ e)› ‹(|⊨∪$\mathcal{G}$ e)›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty
⟨*proof*⟩

interpretation *LA-is-AF-calculus*: *calculus-with-annotated-consrel* ‹{#}› *F-Inf* ‹(|⊨∪$\mathcal{G}$ e)› ‹(⊫∪$\mathcal{G}$ e)›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn
⟨*proof*⟩⟨*proof*⟩
interpretation *core-LA-calculus*: *splitting-calculus* ‹{#}› *F-Inf* ‹(⊫∪$\mathcal{G}$ e)› ‹(⊫∪$\mathcal{G}$ e)›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn
⟨*proof*⟩

notation *LA-is-AF-calculus.AF-entails-sound* (**infix** ‹⊫s∪$\mathcal{G}$ e_{AF}› *50*)
notation *LA-is-AF-calculus.AF-entails* (**infix** ‹⊨∪$\mathcal{G}$ e_{AF}› *50*)

interpretation *AF-calculus-extended* ‹*to-AF* {#}›
core-LA-calculus.core.SInf ‹(⊨∪$\mathcal{G}$ e_{AF})› ‹(⊫s∪$\mathcal{G}$ e_{AF})› *core-LA-calculus.core.SRed$_I$*

core-LA-calculus.core.SRed$_F$ {} {}
⟨*proof*⟩

9.7 Lightweight Avatar

We now augment the earlier calculus into *LA* with the simplification rule BinSplit.

interpretation *with-BinSplit*: *AF-calculus-with-binsplit* ‹*to-AF* {#}› *core-LA-calculus.core.SInf LA-is-AF-calculus.AF-entails LA-is-AF-calculus.AF-entails-sound core-LA-calculus.core.SRed$_I$*

core-LA-calculus.core.SRed$_F$ ‹{}› ‹{}› *core-LA-calculus.bin-splittable*
⟨*proof*⟩

sublocale *LA*: *AF-calculus-extended* ‹*to-AF* {#}›
core-LA-calculus.core.SInf LA-is-AF-calculus.AF-entails LA-is-AF-calculus.AF-entails-sound

core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$ with-BinSplit.Simps-with-BinSplit ‹{}›
⟨*proof*⟩

By Theorem ⟦*annotated-calculus ?bot ?FInf ?entails ?entails-sound ?FRed$_I$?FRed$_F$?fml ?asn*; *Calculi-And-Annotations.statically-complete-calculus ?bot ?FInf ?entails ?FRed$_I$?FRed$_F$*⟧ ⟹ *Calculi-And-Annotations.statically-complete-calculus* (*to-AF ?bot*) (*annotated-calculus.SInf ?bot ?FInf*) (*calculus-with-annotated-consrel.AF-entails ?entails*) (*annotated-calculus.SRed$_I$?bot ?FInf ?FRed$_I$*) (*annotated-calculus.SRed$_F$?FRed$_F$*), we can show that *LA* is statically complete, and therefore dynamically complete by Theorem ⟦*annotated-calculus ?bot ?FInf ?entails ?entails-sound ?FRed$_I$?FRed$_F$?fml ?asn*; *Calculi-And-Annotations.statically-complete-calculus ?bot ?FInf ?entails ?FRed$_I$?FRed$_F$*⟧ ⟹ *Calculi-And-Annotations.dynamically-complete-calculus* (*to-AF ?bot*) (*annotated-calculus.SInf ?bot ?FInf*) (*calculus-with-annotated-consrel.AF-entails ?entails*) (*annotated-calculus.SRed$_I$?bot ?FInf ?FRed$_I$*) (*annotated-calculus.SRed$_F$?FRed$_F$*).

lemma *F-disj-complete*: ‹*statically-complete-calculus* {#} *F-Inf* (⊫∪$\mathcal{G}$ *e*) *F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty*›
⟨*proof*⟩

theorem *strong-static-comp*:
‹*LA-is-AF-calculus.locally-saturated* $\mathcal{N}$ ⟹ $\mathcal{N}$ ⊨∪$\mathcal{G}$ *e$_{AF}$* {*to-AF* {#}} ⟹ *to-AF* {#} ∈ $\mathcal{N}$›
⟨*proof*⟩

sublocale *strong-statically-complete-annotated-calculus* ‹{#}› *F-Inf* (⊫∪$\mathcal{G}$ *e*) (⊨∪$\mathcal{G}$ *e*)
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn core-LA-calculus.core.SInf core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$
⟨*proof*⟩

theorem *strong-dynamic-comp*: ‹*is-derivation core-LA-calculus.core.S-calculus.derive* $\mathcal{N}$ *i* ⟹
LA-is-AF-calculus.locally-fair $\mathcal{N}$ *i* ⟹ *llhd* $\mathcal{N}$ *i* ⊨∪$\mathcal{G}$ *e$_{AF}$* {*to-AF* {#}} ⟹
(∃ *i*. *to-AF* {#} ∈ *llnth* $\mathcal{N}$ *i i*)›
⟨*proof*⟩

sublocale *strong-dynamically-complete-annotated-calculus* ‹{#}› *F-Inf* (⊫∪$\mathcal{G}$ *e*) (⊨∪$\mathcal{G}$ *e*)
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn core-LA-calculus.core.SInf core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$
⟨*proof*⟩

end

end

References

[1] G. Bergeron, F. Krasnopol, and S. Tourret. Formalizing splitting in isabelle/hol. In *ITP (to appear in)*, volume ?? of *LIPIcs*, page ?? Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[2] G. Ebner, J. Blanchette, and S. Tourret. A unifying splitting framework. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 2021.

[3] G. Ebner, J. Blanchette, and S. Tourret. Unifying splitting. *J. Autom. Reason.*, 67(2):16, 2023.